بسم الله الرحمن الرحيم

المحاضرة الأولى:
تاريخ الـ NET . منذ بداية صناعة الحاسوب :

- في بداية الأمر خرج أناس عرفوا بإسم المهندسين والمصممين والعلماء اللذين قاموا بتصميم وهيكلة جهاز الحاسوب وقد نتج عن ذلك جهاز ضخم إحتوى على 5 طوابق تقريباً من النوع الكبير . وكانت فكرة إنشاء جهاز حاسب آلي قد نتجت إثر ضخامة المعلومات وإزالة طريقة الكتابة على الورق وإزالة الوقت الكبير للعمليات الحسابية الأربعة .
- كانت العمليات الأساسية للحاسوب تبرمج عن طريق اللغة الثنائية 0,1 لأن جهاز الحاسوب لم يكن يفهم سوى هذه اللغة .
- ثم ظهرت لغة التجميع المعروفة بالأسمبلي Assembly .
- IBM كانت من الشركات التي في هذا المجال بحيث كانت تصمم جهاز حاسب آلي و تضع فيه نظام تشغيل صغير (O.S) Operating System والتي أدت فيما بعد إلى تطور مصطلح نظام التشغيل O.S والحاجة إلى تطويره .
- ومعنى نظام التشغيل : هو النظام الذي يستخدم أجهزة الهاردوير الموجودة لدى الحاسوب ويتحكم بها بحيث يقوم بإستغلالها للحصول على أكبر قدر ممكن من الكفاءة .
- في ذلك الوقت ظهر طالبين من إحدى الجامعات نمكنا من تطوير وتصميم نظام تشغيل خاص بهما عرف بإسم DOS.
- عندما نظرت شركة IBM إلى النظام الذي صُمم بواسطة هذين الطالبين تبنت نظامهما بالكامل وعملت على تشجيعهما .
- كان النظام يدعى DOS وكلمة DOS جاءت من الجملة المختصرة Disk Operating System ومعناها قرص نظام التشغيل .
- بواسطة لغة الأسمبلي كان العمل صعباً وكانت الجمل طويلة وعدد سطورها طويلة حيث أن المبرمج كان يتعامل مع الذاكرة مباشرة من خلال وضع وجمع وحذف .. الخ ، لذلك فقد كان يتوجب على المبرمج مثلاً لطباعة كلمة أن يكتب 15 أو 20 سطر لتنفيذ ذلك . و نتيجة لذلك كان لابد من إيجاد طريقة جديدة لتوفير الوقت والتقليل من حجم البرنامج فقاموا بإبتكار طريقة جديدة وهي طريقة الدوال Procedure حيث قاموا بوضع أكثر من Instruction بمكان واحد فقط ، ومن هنا نشأ مفهوم المكتبات التي تضم أكثر من دالة وتطورت البرمجة فعرفت بإسم Structured Programming ومعناها البرمجة التركيبية . نتيجة لذلك طُورت عدة لغات مثل COBOL ولغة C الشهيرة و FORTRON والتي عرفت بالجيل الثالث للغات البرمجة .
- كانت شركة أبيل ماكنتوش قد عملت على تطوير نظام فريد من نوعه فقد كان يستخدم الصور للدلالة على الأعمال ، فقامت ببرمجة شكل الملف وشكل المجلد ... إلخ ، وذلك بدل الشاشة السوداء التي كانت معروفة في ذلك الوقت والمسماة Command Prompt وبذلك النظام تم معرفة الـ Interface ومعناها الواجهة والتي كانت User-Friendly حيث كان التفاعل مع المستخدم تفاعلاً مباشراً من خلال الرسومات .
- شركة SUN كانت تمتلك النظام المفتوح Open Source المسمى يونيكس نسبة إلى صانعه وقامت بتشكيل نظام جديد عُرف بإسم لينكس يستخدم الواجهات في عمله .
- ظهرت في هذه اللحظة التسابق لإنتاج نظام تشغيل كامل من واسطة كبريات الشركات مثل Microsoft وشركة Sun وشركة Apple Macintosh و باقي الشركات الكبرى .
- وأيضاً في هذه الأثناء وبينما كان هناك تقدم وتطور من قِبل شركات Sun وشركة Apple Macintosh كانت شركة Microsoft تعيش ركود نسبي لأنها كانت تعتمد نظام الـ Command Prompt في تلك الأثناء .
- شعرت شركة مايكروسوفت بعجزها وبتنحيها عن السوق فقامت بطرح تطبيق Application وليس نظام تشغيل يعتمد الواجهة الرسومية والذي عرف بنظام Windows 3.X .
- ظهرت الحاجة لتوحيد طريقة البرمجة لدى المبرمجين لكي يكون الكود ديناميكي وسلس وله قواعد وشروط ، ولكي لا يخرج كل مصمم بأفكار بعيدة عن الهدف المقصود فظهر مفهوم الـ Object Oriented Programming والمعروف بالرمز (OOP) وأدت إلى ظهور العديد من اللغات ومن الأمثلة عليها ... , Small Talk , C++ بما عرفت بالجيل الرابع للغات البرمجة .
- بواسطة البرمجة الموجهة بالأهداف أنتجت شركة مايكروسوفت شبه نظام وسموه Windows 95 وسبب أنه كان شبه نظام أنه كان يعتمد على نظام الـ DOS في أداءه وعمله .

- وظهرت الحاجة إلى إخراج جيل من البرمجيات المتكاملة من خلال شركة مايكروسوفت والتي سميت فيما بعد بـ Microsoft Office .
- وبعد سنتين تقريباً طرحت شركة مايكروسوفت نظام تشغيل وسمته Windows 97 ولكنه فشل فشلاً سريعاً فقد طرح وخلال شهرين فقط تم سحبه من الأسواق لرداءته .
- ولكنها سرعان ما قدمت أول نظام تشغيل مستقل بواجهة رسومية والذي عرف بإسم Windows 98 ، و هنا ظهرت الحاجة لإنشاء برمجيات متكاملة على طريقة الـ Enterprise التي تعني وجود جميع الحلول في مجموعة واحدة .
- قامت شركة مايكروسوفت بالتفكير بفكرة لإجبار جميع العملاء على إستخدام نظامها التشغيلي ، فقد قامت بطرح معالج لبعض اللغات مثل لغة الـ C ولغة الـ BASIC ، و قامت بالتسهيل على المبرمجين الحصول وبالمجان على الـ API المستخدمة من قِبل النظام والمختصرة من جملة Application Program Interface والتي تعني واجهة البرامج التطبيقية . فقامت بالتسهيل على المبرمج ، فمثلاً إذا أراد المبرمج كتابة كود لإظهار نافذة فقط فإنه سيبقى وقت طويل جداً وسيكتب كود طويل جداً ، فقامت مايكروسوفت بطرح المكتبات لهذه الغاية بحيث ينتج نافذة كما في نظام التشغيل بوقت وكلفة قليلين جداً . وبذلك جذبت العديد من المبرمجين و العملاء إليها بواسطة هذه الطريقة . وبالطبع فإن البرنامج الناتج من هذه العملية لن يشتغل سوى على نظام التشغيل Windows .
- قام المبرمجين بكتابة البرامج المختلفة بواسطة نظام API لأنه كان سلس وسهل الإستعمال ونتيجة لذلك فقد زادت مبيعات شركة مايكروسوفت أضعافاً مضاعفة والتي وصلت سنوياً إلى 3.5 مليار دولار سنوياً .
- في هذه الأثناء قامت شركة SUN بطرح لغة جديدة من إنتاجها والتي عرفت بإسم جافا JAVA لملاحقة شركة مايكروسوفت وبذلك تكون هذه الشركة قد قطعت شوطاً كبيراً بهذا المجال .
- ظهرت مكتبتان هنا من إنتاج الشركتان وهما :
  SDK : من شركة مايكروسوفت وهي إختصار لجملة Software Developer Kit .
  JDK : من شركة SUN وهي إختصار لجملة JAVA Developer Kit .
- لكن يوجد جانب لم تعيهما تلك الشركتان وهو قواعد البيانات Database والتي أصبحت حالياً الجانب المهم والأكبر في إستخدام الحاسب الآلي فقد كانت شركة Oracle الرائدة في هذا المجال والسباقة إليه . وظهرت شركات أخرى مثل شركة Fox Pro التي أيضاً كانت من الشركات الكبيرة في هذا المجال .
- تنبهت شركة مايكروسوفت باكراً إلى خطورة هذا الجانب فقامت بشراء ملكية شركة Fox Pro وضمتها إلى أملاكها وقامت لاحقاً بإنشاء المكتبة الخاصة للـ Database التي عرفت بإسم SQL .
- و بإستخدام قواعد البيانات أنتجت شركة مايكروسوفت لغات جديدة مثل Visual C++ , Visual Basic , Visual Fox Pro وضمتها إلى قائمتها البرمجية .
- و بإستخدام تكنولوجيا الإنترنت فقد قامت شركة مايكروسوفت بطرح لغة خاصة لمعالجة صفحات الإنترنت والي سُميت فيما بعد بالإسم InterDev وهي عبارة عن صفحات إنترنت نشطة ذات الإمتداد المعروف ASP .
- كانت شركة SUN تعمل على تحديث وتطوير لغتها (JAVA) أول بأول من حيث قواعد البيانات و التعامل مع الإنترنت .
- جاءت فكرة الـ Script والتي تعني جزء من كود لغات الإنترنت في صفحة HTML .
- بسبب الطمع والجشع الذي كان عند شركة مايكروسوفت فقد قامت بالخطوة القاتلة وهي أنها أنشأت لغة جديدة وسمتها بإسم J++ Visual ، ولكنها لم تسلم بتلك الفعلة فقد قامت شركة SUN برفع دعوة قضائية على شركة مايكروسوفت ولأنها صاحبة اللغة فقد خسرت شركة مايكروسوفت تلك القضية وتم تغريمها مبالغ طائلة وصلت إلى 3 مليار دولار أمريكي ومُنعت من إستعمالها والتطوير عليها لذلك السبب .
- يمكن تلخيص الإصدارات للغات البرمجة في تلك الفترة لشركة مايكروسوفت بالجدول التالي :

Visual Studio 2	Visual Studio 4	Visual Studio 5	Visual Studio 6
C	Visual C++	Visual C++	Visual C++
Basic	Visual Basic	Visual Basic	Visual Basic
	Visual Fox Pro	Visual Fox Pro	Visual Fox Pro

		InterDev	InterDev
			Visual J++

- ظهرت مشاكل عديدة في لغات شركة مايكروسوفت فمثلاً وجود أوامر في لغة السي ++ وعدم وجودها في الفيجوال بيسك وكذلك فقد كان من الصعب تشغيل برنامج على الفيجوال بينما هو مكتوب بلغة السي ++ ، في هذه الأثناء كانت شركة SUN تسير على الطريق الصحيح وتعمل على تطوير لغتها أولاً بأول . ومن المشاكل الأخرى لمايكروسوفت أن مبرمج اللغة الواحدة لا يستطيع أن يتعامل أكثر من نسخة من برامج قواعد البيانات فمثلاً كل مبرمج واحد يستطيع التعامل مع لغة واحدة لا يسعه إستعمال غير مكتبة واحدة فقط من المكتبات التالية ODBC , ADO , DAO ... الخ ، وكل مكتبة متخصصة بلغة واحدة لا يستطيع المبرمج إستعمالها في لغة أخرى وفي هذا تقسيم المبرمجين إلى أقسام عديدة وكل مبرمج له مجال يختلف عن مجال زميله في المهنة ، بينما كانت شركة SUN تمشي بدون أي مشاكل فهناك لغة واحدة فقط وهي الجافا وهناك مكتبة واحدة وموحدة فقط لقواعد البيانات وهي JDBC .
- نظرت شركة مايكروسوفت بنظرة عميقة للموضوع وأسفر عن ذلك خطة جديدة مع عام 2000 وهي إخراج لغات موحدة لها نفس مكتبات التشغيل ولها إطار واحد فقط Frame Ware وأدت إلى ظهور لغات جديدة عرفت بمجموعة الدوت نيت NET . فقدمت لغات جديدة مثل VC# وضمتها في مجموعة NET . Visual Studio 7 وضمت :

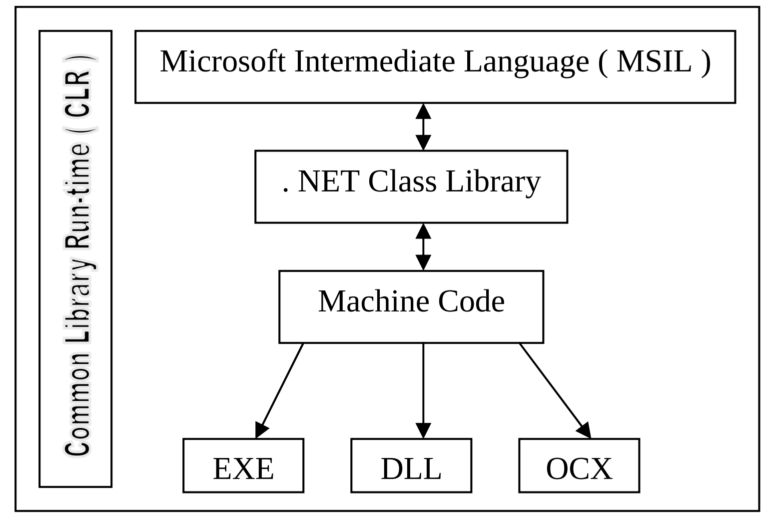
1 – Visual C # (2000)
2 – Visual C ++ (2002)
3 – Visual Basic (2002)
4 – Visual J# (2003)

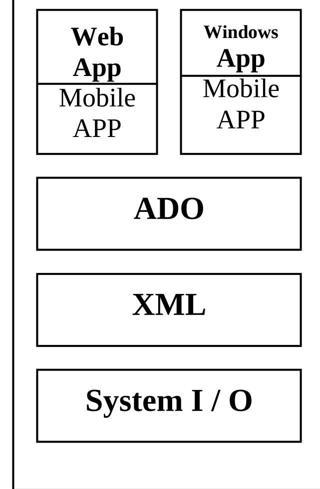
في هذه النسخة قاموا بضم لغة الـ J# ولعلك تتساءل كيف ذلك ؟؟ والمحاكمة ؟؟
هنا قامت شركة مايكروسوفت بتصميم تلك اللغة من الصفر وقاموا بتطويرها لتحاكي لغة الجافا .

كل اللغات السابقة كانت قادرة وبكفاءة التعامل مع الـ ADO , XML و بإستطاعة المبرمج تصميم وصناعة البرامج في المجالات التالية :

Windows Application , Web Application , Mobile Application , Smart Device …

و قاموا بإنشاء Compiler مترجم لكل لغة وهو موحد ويستدعي مكتبة واحدة كما في الشكل التالي :





Jordan , Amman Tel : (00 962 79 6527425) Omssd84@hotmail.com


بسم الله الرحمن الرحيم

المحاضرة الثانية :

مفهوم وفلسفة البرمجة هدفية الغرض (الكائنية) (Object Oriented (OOP :

- كما قلنا وتحدثنا سابقاً في المحاضرة الأولى من أنه لا بد من وجود معيار ومقياس لبرمجة وكتابة الكود من طرف المبرمجين ولنخرج جيل ذو خبرة عالية و مرونة وسهولة . وبسبب هذه الفلسفة ظهر ما يعرف ببرمجة الكائنات الموجهة (Object Oriented (OOP والتي تستند إلى إستخدام المكتبات Library و إستخدام الدوال Function والتي عرفناها قبل ذلك بالبرمجة التركيبية Structured Programming والتي ظهر مفهوم البرمجة بواسطة الكائنات الموجهة على إثرها .
- والآن ما معنى البرمجة الموجهة بالكائنات وكيف نستطيع الحكم على لغة بأنها OOP أو لا ؟

- قبل أن أبدأ أريد أن أسرد لك طرفة سريعة حدثت مع معلمي الأستاذ ((معن الجيتاوي)) فعندما كان يشتغل في شركة WinWord الموجودة هنا في الأردن والتي تعتبر رقم 1 في الشركات البرمجية والتي بواسطتها تم إدخال علم الحاسوب ونظم المعلومات الحاسوبية في المدارس والجامعات . وبصفته Project Manager في تلك الشركة جاءه شخص من شركة IBM الرائدة في عالم الحاسوب . وأراد أن يسأله ويختبر معلوماته وكونه ذو الخبرة العالية في السي ++ وهو في المقابلة فقام بسؤاله السؤال التالي :
هل لغة ++C تعتبر Full OOP أم لا ؟ قال له لا تعتبر بشكل كامل ولكنها تدعمها ، مع العلم أن الجميع يقولون بنعم ((تعتبر OOP بدعم كامل)) فتنبه هذا الشخص إلى ذكاء أستاذي ومعلمي فقال له إنتهت المقابلة بنجاح .

والسبب في ذلك ((لغة الـ ++C)) ليست لغة برمجة موجهة بالكائنات التالي :
فلكي نتكلم عن لغة أنها Full OOP يجب أن تتوافر فيها الشروط الأربعة التالية :
1 – Encapsulation
2 – Abstraction
3 – Inheritance
4 – Polymorphizen

ولنأتي الى الشروط الأربعة بالتفصيل الممل :

الشرط الأول : Encapsulation

ومعناه هو تجميع أشياء ووضعها كاملة بمكان ما . ومن إسمها تستطيع تمييز ذلك فمعناها الكبسولة ولتعريفها جزأين :
الجزء الأول : هي تعريف مكان ((Class)) ونضع فيه كل المتغيرات والدوال التي لها علاقة ببعضها ومن ثم نسند إليه إسماً
الجزء الثاني : هذا المكان لا يستطيع أحد الدخول إليه إلا بصلاحيات تحدد من قِبل المبرمج فيوجد قسمان رئيسيان لها وهما الـ Public وهو القسم العام والـ Private وهو القسم الخاص .

الشرط الثاني : Abstraction

وهي إضافة دالة عامة Function داخل الـ (Class) لا أقوم بتحديد التفصيلات لها إلا في قسم البرنامج الرئيسي .
فمثلاً :
نريد صنع عدة كائنات ((Class)) أو Modules في جامعة معينة فينتج من ذلك كائن للدكتور وكائن للطالب وكائن للمدير وكائن للعامل و ... الخ وأردنا أن نجري بحث فيه من خلال البرنامج الرئيسي فأتت مجموعتان أول مجموعة قامت بتعريف 100 متغير و100 دالة لهذا الغرض فكتبت في كل كائن دالة للبحث في خاصية معينة فمثلاً دالة لبحث حسب الإسم ودالة أخرى لبحث الرقم ودالة أخرى لبحث الرقم و الإسم معاً وهنا تكون المجموعة الأولى قد أطالت من برنامجها وقللت من كفاءة البرنامج التي صنعته . وتأتي المجموعة الثانية فتستخدم 30 متغير و 30 دالة وتؤدي نفس الغرض فهي هنا إستخدمت دالة خاصة بالبحث في كل كائن بدلاً من عدة دوال وهي هنا لم تحدد البحث بواسطة ماذا ولكنها عند تشغيل البرنامج تقوم وبناءاً على طلب المستخدم بالبحث عن أي خاصية يريدها .



وكمثال ثانٍ :
نفترض أن عندنا شركة مشروبات غازية تقوم بإضافة علبة خامة ((قياسية)) فتأتي شركة أ وتطلب مئة عبوة هنا تستخدم الشركة العبوة الخامة وتقوم بالتعديل وإضافة الإعلانات الخاصة بشركة أ على العلب المطلوبة وتأتي شركة ب وتطلب 300 عبوة فتصنع كما صنعت مع شركة أ .
ولنتوقف عند هذا المثال قليلاً ونسأل أنفسنا : ماذا لو قمنا بالأصل بتصميم العلب للشركة أ وقامت الشركة ب بالطلب ؟؟
ستحدث خسارة في الوقت والجهد ، فهنا يكون معنى الـ Abstraction قد توضح لنا ورسم في مخيلتنا .

الشرط الثالث : Inheritance
لو سألنا أنفسنا ما معنى هذه الكلمة لأجبنا أنفسنا بالسرعة معناها التوارث ((طيب)) وماذا يعني ذلك ؟؟!!

العديد من المبرمجين يفهموا هذا المفهوم فهماً خاطئاً كالتالي :
إذا كان عندنا أربع أو خمس كائنات تحتوي جميعها على خاصية موحدة أو خاصيتان أو عدة خواص فإننا نقوم بإنشاء كائن جديد نضع فيه الخواص المتشابهة في كل كائن ونقول لأنفسنا أن جميع الكائنات الأخرى هي عبارة عن Inheritance أي متوارثة من الكائن الأصلي ((الجديد)) وهذا بمفهومه المنطقي صحيح ولا غبار عليه . ولكن النظرة من الجانب العملي لها وجهة خاصة فعندما نقوم بتصفية الخواص المشتركة ووضعها في كائن جديد لم يعد للكائن الجديد معنى . وإليك المثال التالي لتسهيل الفهم :
لنفترض أننا نمتلك الكائنات التالية : الطالب و الدكتور والعامل والسكرتير و ... الخ
كل الكائنات السابقة لها خاصيتان متشابهتان وهما الإسم والرقم .
فلو قمنا بفصل هذين الخاصيتين وجعلناها في كائن واحد (إنسان) ثم قمنا بإعطائه دور الأب وإعطاءه دور الإبن ، فهنا يمكننا القول بأن الكائنات الأخرى هي Inheritance أي متوارثة من الكائن الأصلي ((الإنسان)) وهذا كما قلت منطقياً مقبول . ولكن !!!
ماذا لو أخذنا الكائن الجديد على حدة وسألنا أنفسنا ماذا يمثل ؟؟!! لا شيء وليس له معنى لأنه عندما كانت هذين الخاصيتين في كائن الطالب كانت هناك تفاصيل أكثر دقة تفسر الكائن وتدل على أنه كائن الطالب . وبعد أخذهما على حده فلا يمكننا التمييز ما هو ؟؟!! .

سؤال يجب طرحه هنا : ما الفرق بين الـ ((Class)) الكائن وبين الـ ((Object)) ؟؟!!
سؤال محير أليس كذلك !! إليك الفرق :
الـ Class هو شيء نظري غير موجود على الواقع فهو عبارة عن تعريفات لشيء معين نظرياً .
الـ Object وهو الكائن النظري عند تطبيقه على أرض الواقع ليصبح عنصراً وكائناً عملياً .

الشرط الرابع : Polymorphizen
يمكن تقسيمه إلى نوعين وهما الـ Overload والـ Override . وإليك تبييناً لكلاهما :
Overload : وهي عبارة عن دوال لهم نفس الإسم ولكن تختلف في عدد الباراميتر ((الوسائط التي تأخذها)) أو نوعها
Overload Operator : فمثلاً نريد إشارة الزائد + في المعادلة التالية 1+1=2 أن نحوله إلى حرف وصل ((&)) فبإمكاننا تعريف نفس الإسم ولكن أغير في النوع والكود لتصبح المعادلة التالية 1 & 1 = 11 .
Override : وهي عبارة عن كتابة نفس الدالة الأصلية طبق الأصل عنها ولكن أغير في الكود فمثلاً نعرف الجملة التالية X = 3 ثم نقوم بعدها بتعريف الجملة التالية X = 7 فيقوم الحاسوب بحذف الأولى وتطبيق الثانية . هذا بالنسبة للمتغيرات
أما بالنسبة للدوال فإذا عرفنا دالة في كائن معين ثم قمنا بإعادة كتابة الدالة ولكن غيرنا في الكود فيقوم الحاسوب بحذف الأولى و إعتماد الثانية .

إذن يحق لي كمبرمج إستعمال الدالة Function أكثر من مرة بإضافة تغييرات في النوع أو البارامتير أو بإعادة تعريفه .

إذن :
Overload : تغيير النوع أو عدد البارامتير لدالة معينة .
Override : تغيير الكود فقط ولا تغيير على النوع أو البارامتير .

بسم الله الرحمن الرحيم

المحاضرة الثالثة :

بيئة الفيجوال ستوديو دوت نيت – قسم الـ Console :

- قبل التحدث والدخول إلى هذه البيئة يجب أن تمتلك نسخه منها وعنوانها :
Microsoft Visual Studio .NET 2003 وهي تتكون من 6 سيديات على الأغلب وتتكون من :

2 سي دي للدوت نيت 2003
3 سي دي للمكتبة MSDN
1 سي دي Component وهو عبارة عن مكونات ما قبل البداية
1 سي ي SQL 2000 Server وهو غالباً لا يكون موجود معها ((إضافي)) للتعامل مع قواعد البيانات

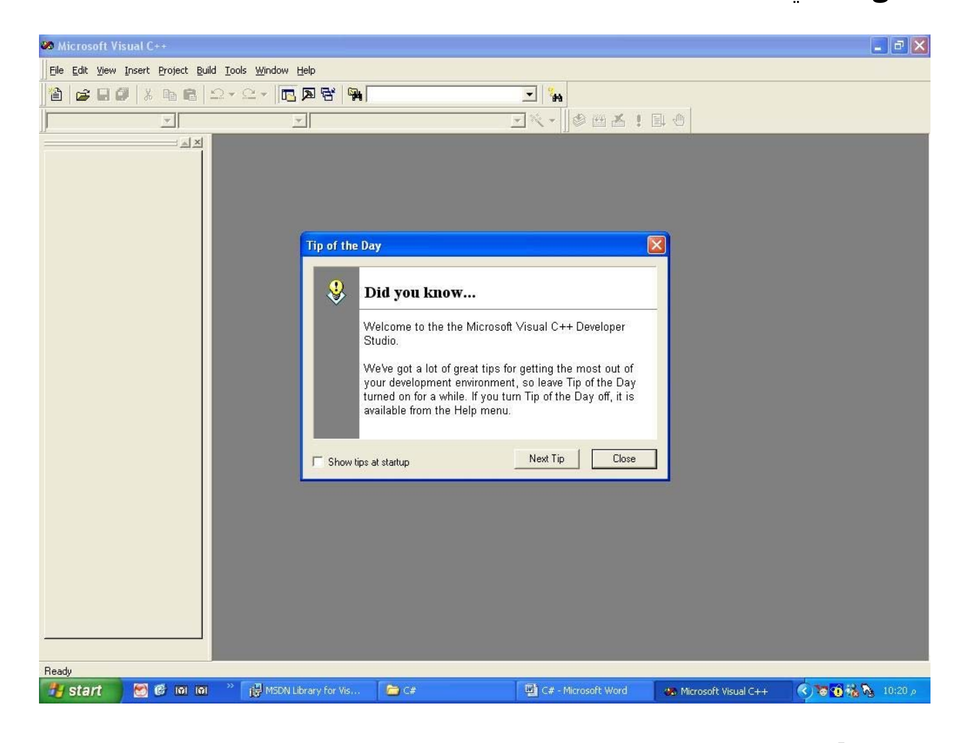
لنبدأ الآن :
إذا نظرنا إلى الإصدارات القديمة من مكتبة الـ Microsoft Visual Studio مثل الإصدارة السادسة أو الخامسة وأخذنا على سبيل المقارنة برنامج الـ Visual C++ و برنامج الـ Visual Basic وقارناهم كالتالي :

من الناحية النظرية :
إذا قمنا بالعمل والتصميم على برنامج السي ++ ثم فجأة إنتقلنا إلى واجهة الفيجوال بيسك سنرى هناك إختلافاً كبيراً بينهما . وهذه نقطة ضعف لدى مايكروسوفت . بينما لو نظرنا إلى مجموعة الأوفيس 2003 بالعمل على وورد ثم الإنتقال إلى بوربوينت مثلاً سنجد هناك تشابهاً بنسبة 95%
إذا فشل الإصدار السادس من الناحية النظرية .

أما من الناحية العملية :
إليك التالي :

- قم بالضغط على
Start → Program File → Microsoft Visual Studio 6.0 → Microsoft Visual C++ 6.0

سيظهر لك الشكل التالي :

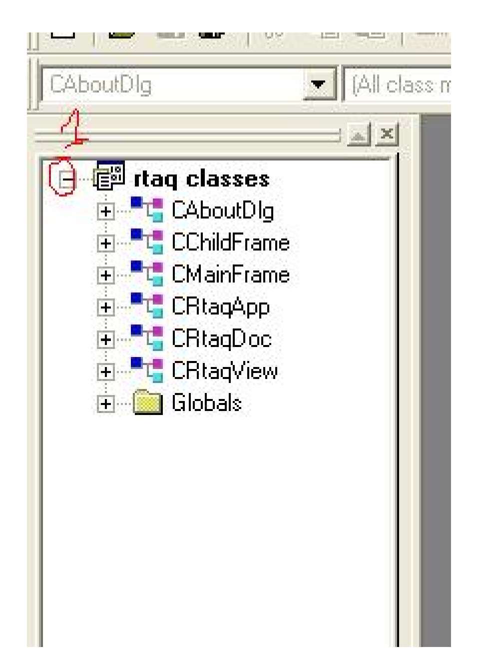


قم بالضغط على Close ثم قم بإختيار

File → New → Project → MFC AppWizard (exe)

كالصورة التالية :

وقم بإختيار الإسم لها كما في الرقم 4 و 5 ثم إضغط على موافق .
وسيظهر لك شاشة أخرى قم بإختيار Finish منها ثم OK قم بفتح الشجرة من قسم نافذة المشروع كالتالي :



أنظر إلى عدد الـ Classes الموجودة هنا ستجدها 6 حسناً خزن هذا الرقم في مخيلتك

الآن إنتقل إلى فيجوال بيسك حسب التسلسل التالي :
Start → Program File → Microsoft Visual Studio 6.0 → Microsoft Visual Basic 6.0

سيظهر لك الشكل التالي :

لاحظ أننا إخترنا المشروع التنفيذي كما في السي ++ ولكن هنا إنظر إلى الرقم 3 ماذا تلاحظ ؟؟
وجود ملف واحد فقط ((طيب في السي ++ موجود 6)) إذاً فشل الإصدار السادس كذلك في الإختبار من الناحية العملية لأنه يجب على كل لغة في الإصدار أن تحتوي على نفس عدد الملفات في كل مشروع مشابه

الآن وبعدما لاحظنا المقارنة في الإصدار السادس ننتقل إلى مجموعة الدوت نيت ((2003)) ونلاحظ الفرق :
قم بتتبع التسلسل التالي :
Start → Program File → Microsoft Visual Studio .NET 2003
→ Microsoft Visual Studio .NET 2003

ماذا تلاحظ من المسار ؟؟
أنه لا يوجد تقسيمات للبرامج كما في الإصدار السادس يعني أنه لا يوجد سوى بيئة واحدة فقط لكل اللغات المضمنة في تلك اللغة . إذن من الناحية النظرية تمام .

بعد أن تتبعت المسار ستظهر لك النافذة الرئيسية كالتالي :



لاحظ معي وبالترتيب لكي تتعرف على واجهة البرنامج :

1 – وهو شريط العنوان والكل يعرفه .
2 – شريط القوائم و به أوامر للمساعدة في بناء المشروع في بيئة الدوت نيت
3 – شريط الإختصارات و به أوامر مختصرة ((كثيرة الإستعمال)) من شريط القوائم ((2))
4 – صفحة البداية وهي الصفحة الرئيسية لهذه البيئة وتتكون من :

- المشاريع Projects ((6)) وتتكون من التالي :
  - نافذة المشاريع وتحتوي على آخر أربع مشاريع ((رقم 7)) قمت بالعمل بهم مع التاريخ حيث يكتب التاريخ إما اليوم أو الأمس أو تاريخ العمل به ((التعديل عليه)) .
  - أزرار إنشاء أو فتح مشاريع سابقة ((12)) ((13)) على التوالي .

- المصادر من الإنترنت Online Resource :
  والمهمة لهذه الصفحة الحصول على الأمثلة أو طرح المشاكل التي تواجهك في منتديات البرمجة لموقع مايكروسوفت للحلول المباشرة وهذه الخدمة متوفرة شريطة الربط مع الإنترنت .

- ملفك الشخصي My Profile :
  هنا يمكنك التعديل على الواجهة بشكل عام أو إختصارات لوحة المفاتيح كالتالي :
  لو فتحت هذه النافذة لوجدت الشاشة كالتالي :





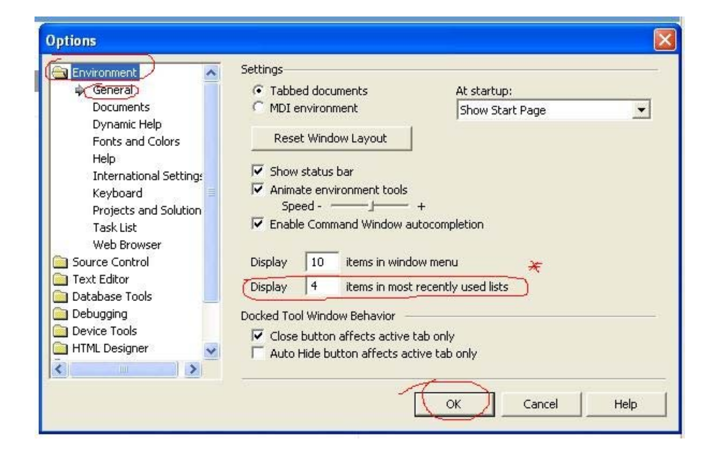
((3)) يمكنك إختيار أي إختصارات للغة قد تعودت عليها من قبل كالسي ++ 6
((4)) يمكنك تغيير الشكل العام أو الإطار العام لبيئة الدوت نيت
((5)) يمكنك أن تختار طريقة عرض المساعدة للغة قد تعودت عليها من قبل
* كل الأرقام السابقة يمكن أن تعطيك ملف خاص كما في ((2))
و الإختيار السادس هو إجراء حدث عند بداية تنفيذ بيئة الدوت نيت .

5 - شريط لعرض النوافذ الموجودة أمامك فإذا إختفت واحدة ما عليك سوى الضغط على إسمها وستظهر لك حالاً .
9 – شريط الأدوات ولكن في السي شارب Console لن نستعمله فهو خاص بتطبيقات النوافذ .
10 – نافذة ملفاتك في مشروعك الذي تعمل عليه الآن .
11 – نافذة المساعدة .
14 – في كل نافذة ستجد إشارة الدبوس هذه ومعناها أخفي تلقائياً بمعنى أنه إذا ذهبت الماوس من فوق تلك النافذة قم بإخفائها تلقائية . وإذا إقتربت منها قم بإظهارها تلقائياً .

ملاحظة (1) :
في القسم رقم ((6)) من الصورة السابقة يمكنك زيادة عدد المشاريع عن أربعة وذلك بالمسار التالي :

Tools → Option → Environment → General → Display recently

كالصورة التالية :



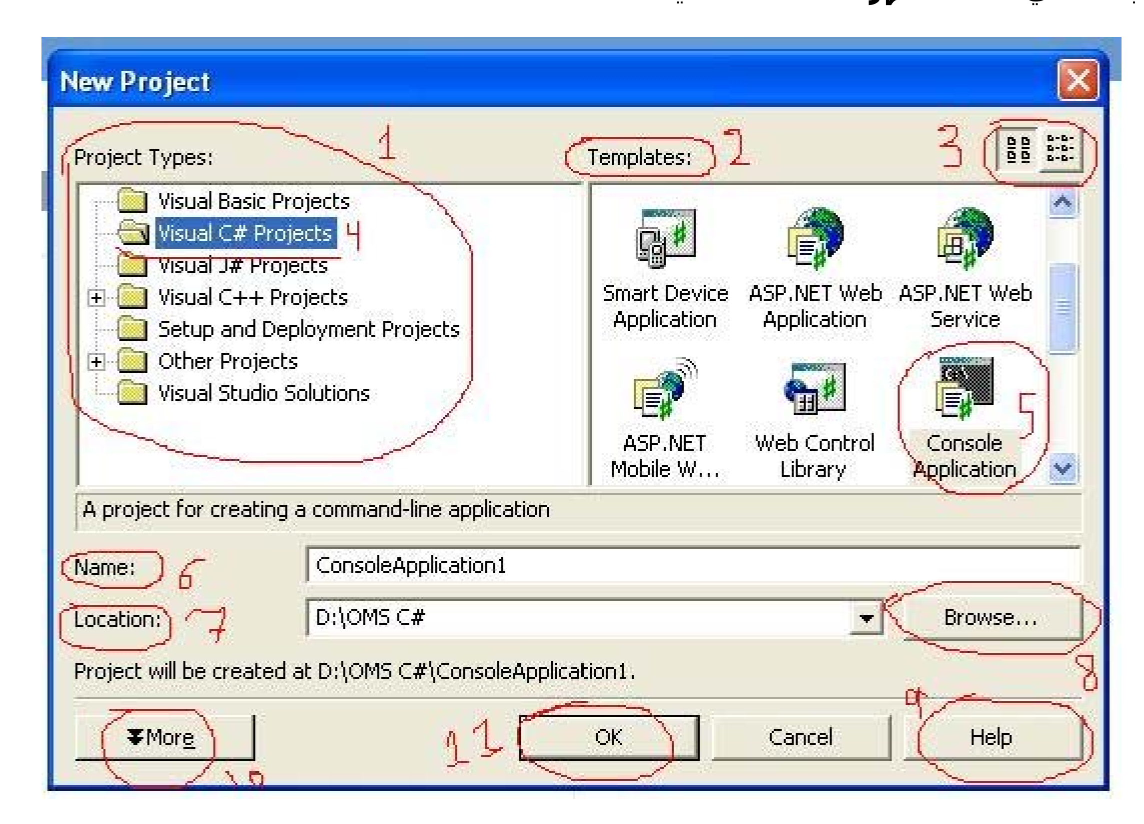


فقم بزيادة العدد عند إشارة النجمة ((*)) ثم قم بالنقر على موافق

ملاحظة (2) :
إذا بدأت البيئة ثم لم تظهر لك نافذة البداية ((صفحة البداية)) إذهب إلى :
Help → Show Start Page

الآن وبعد ما تعرفت على أقسام النافذة الرئيسية لنبدأ بمثالنا الأول بالـ Console Application :

إذهب الآن إلى File → New → Project .
أو قم بالضغط على Ctrl + Shift + N .
أو قم بالنقر على زر المشروع الجديد من شريط الإختصارات .
إذا نجحت في ذلك ستظهر لك النافذة التالية :



لنقف قليلاً عند هذه النافذة لكي نبين أجزائها :
1 – أنظر هنا في هذا القسم ((1)) ستلاحظ أسماء جميع اللغات المستعملة في بيئة الدوت نيت وبهذا تكون شركة مايكروسوفت قد نجحت في تجميع جميع اللغات في إطار واحد .
2 – تجد جميع الأقسام ((القوالب)) التي يمكنك تصميمها مثل تطبيقات الويندوز وتطبيقات الكونسول وتطبيقات السمارت ديفايس ((Pocket PC)) وذلك في القسم ((2)) .
3 – يمكنك أن تصغر وتكبر الأيقونات الموجودة في القسم ((2)) بواسطة الأزرار في القسم ((3)) .
4 – إذن الآن لنقم بفتح مشروع سي شارب فقم بالضغط على ((4)) .
5 – ثم بعدها قم بالضغط على ((5)) ولاحظ أن تطبيقات الـ Console سبق لنا تعريفها وهي التطبيقات التي تفتح شاشة سوداء كنظام الـ DOS .
6 – يمكنك تسمية المشروع من الرقم ((6)) وتذكر أنه سيسمي الملفات كلها بهذا الإسم ((ملفات العمل)) .
7 – من إسمها ((7)) تعرف أنها مكان منطقة العمل Directory . وبإمكانك تحديد مكان معين بالضغط على الزر الذي يحمل الرقم ((8)) .
8 – إذا ضغط هذا المفتاح ((9))سيفتح لك نافذة مساعدة عن هذه الصفحة فقط .
9 – بإمكانك زيادة الخيارات المستخدمة عندك بإضافة عملك في مجلد جديد على المسار الذي حددته في الخطوة رقم ((7)) وذلك بالضغط على الرقم ((10)) والذي يحمل كلمة More .
10 - وأخيراً قم بعمل OK ((موافق)) لنبدأ مشروعنا اليوم .

إذا نجحت في تطبيق السابق ستظهر معك النافذة التالية :

First Application With Console - Microsoft Visual C# .NET [design] - Class1.cs

```
using System;

namespace First_Application_With_Console
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Class1
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            //
            // TODO: Add code to start application here
            //
        }
    }
}
```

Solution Explorer - First Applica...
Solution 'First Application With Console
First Application With Console
References
App.ico
AssemblyInfo.cs
Class1.cs

أنظر إلى النافذة السابقة ولاحظ معي :

1 – قام بفتح نافذة جديدة ((1)) وسماها Class1.cs .

يتكون الإسم من قسمين كالتالي :

القسم الأول يحمل إسم Class1 وهذه تعتبر الكائن الرئيسي في المشروع لأنه سبق لنا أن قلنا أننا نسعى للغة تدعم البرمجة الكائنية OOP ولأننا قلنا أنه يجب أن يكون هناك على الأقل كائن واحد وهو يحمل الدالة الرئيسية Main Function وبهذه الخطوة إرتقت مايكروسوفت إلى مستوى البرمجة بالكائنات والتي لم تكن موجودة بالإصدارات السابقة .

والقسم الثاني يحمل إسم الإمتداد cs ومعناه C Sharp أي إسم اللغة

2 –في القسم ((2)) ماذا تلاحظ ؟

المشروع يحتوي على 3 ملفات فقط . ملف للأيقونة الناتجة في المشروع وملف التحويل للغة الأسمبلي والملف الثالث المحتوي على الكود الذي نقوم بكتابته .

ملاحظة سريعة هنا :

حاول و إفتح مشاريع غير السي شارب ((ماذا تلاحظ)) ؟

ستقول لي كذلك تفتح 3 ملفات . وفي هذه الخطوة إستطاعت شركة مايكروسوفت توحيد أنماط جميع لغاتها ضمن باقة دوت نيت .

Jordan , Amman Tel : (00 962 79 6527425) Omssd84@hotmail.com



- لنأتي الآن إلى منطقة العمل ونشرحها بالتفصيل :
لاحظ معي النافذة كالتالي :

```
1   using System;
2
3   namespace First_Application_With_Console
4   {
5       /// <summary>
6       /// Summary description for Class1.
7       /// </summary>
8       class Class1
9       {
10          /// <summary>
11          /// The main entry point for the application.
12          /// </summary>
13          [STAThread]
14          static void Main(string[] args)
15          {
16              //
17              // TODO: Add code to start application here
18              //
19          }
20      }
21  }
22
```

((إقرأ الملاحظة بعد الموضوع مباشرة))

في السطر رقم (1) جملة `using System;` نستطيع إستنتاج التالي :
* جميع الكلمات الموجودة في منطقة العمل والتي تحمل اللون الأزرق هي كلمات محجوزة Keyword لا نستطيع إستعمالها كمتغيرات .
* تعتبر لغة السي شارب لغة حساسة Case Sensitive يعني أن المتغيرات (Age, AGE, aGE , AgE) كلها متغيرات لا يشبه بعضها بعضاً وتعاملها هذه اللغة كل واحدة على حده .
* نهاية كل جملة تحتوي على فاصلة منقوطة ; وهي تعبر عن نهاية السطر .

ووظيفة السطر الأول هي إستدعاء مكتبة ((سنتفق على تسميتها `namespace`)) للتعامل مع المشروع بشكل جيد مثل جمل الإدخال والإخراج وتعادل هذه الجملة بكلمة #include في لغة السي هنا إستدعى مكتبة الـ System ولاحظ أن أول حرف كبير وهذه المكتبة مختصة بالدوال الرئيسية التي تستخدم بكثرة كجمل الإدخال والإخراج وتعادل هذه المكتبة مكتبة الـ iostream.h المستخدمة في لغة السي .

في السطر رقم (3) جملة `namespace First_Application_With_Console` نستطيع إستنتاج التالي :
* قام بإنشاء مكتبة تحتوي على المشروع الذي نكتبه الآن .
* قام بوضع علامة تحت السطر (_) بدلاً من الفراغات والتي أصلاً إسم مشروعنا الحالي .
* تلاحظ أنه يوجد مربع صغير يحتوي على إشارة ناقص (-) ووظيفته إخفاء تفاصيل الكلاس أو الدالة المشار إليها وبعد الضغط عليه يظهر لنا مستطيل يحتوي على ثلاث نقاط (...) إذا حركت الماوس عليه يعطيك محتوى الكلاس أو الدالة المشار إليها كاملاً كشكل ملاحظة Tag بمستطيل أصفر اللون فيعرض لك محتوياتها مهما بلغت من الطول .
والهدف من هذا المربع هو إخفاء دالة أو كلاس سبق لنا أن كتبناها ولا نريد إظهارها .

ومعنى هذه الجملة أنه قام بإنشاء مكتبة خاصة والتي تحتوي على الـ Classes الموجودة في مشروعنا الحالي فمثلاً إذا أردنا إستدعاد دالة من الدوال في مشروع آخر ما علينا سوى كتابة إسم المشروع الحالي ثم إتباعه بنقطة ثم إسم الكلاس أو الدالة التي نريد إستعمالها .

في السطر رقم (4) القوس المشهور `{`) والذي يدل على بداية الدالة أو الكلاس وطبعاً نغلقها بالمثل بإستخدام القوس المثيل `}`) كما في السطر رقم (21) .

في السطر رقم (5) `/// <summary>` جملة تعيق ولكنها للغة الـ XML دعها جانباً لن تفيدنا الأن في الوقت الحالي فلها وقتها . لاحظ أنها تحتوي على ثلاث أقواس .

في السطر رقم (7) لاحظ وجود إشارة (-) على العمود وهذا يعني بداية الجملة الأولى في الكلاس أو الدالة Function .

في السطر رقم (8) تلاحظ وجود إسم الكلاس المستعملة في مشروعنا الحالي .

في السطر رقم (13) `[STAThread]` أي كلمة موجودة بين قوسين ((مربعين)) تسمى خاصية Attribute وسنقوم بشرحها لاحقاً .

في السطر رقم (14) `static void Main(string[] args)` هنا توجد الدالة الرئيسية لمشروعنا .

والجملة تتكون من :

* `static void` تحديد نوع الدالة الرئيسية فهي من نوع Void التي تعني أن الدالة لا ترجع أي قيمة وهي من القسم Static من النوع الإستاتيكي .

والنوع الإستاتيكي يمكن شرحه كالتالي : لو أنك عرّفت متغير ما بالنوع الإستاتيكي في دالة معينة في داخل كلاس معين ثم إستدعيت الدالة وأجريت تعديلات على هذا المتغير وخرجت من الدالة فإن الأصل أن يحذف المتغير من الذاكرة ولكن المتغير الإستاتيكي يقوم بتسجيل نفسه في الذاكرة ما دام البرنامج أو المشروح الذي صممته في وقت التنفيذ . فمثلاً لو عرفت في دالة معينة المتغير X من نوع Int من النوع الإستاتيكي وقمت في سطر تالي بزيادة هذا المتغير بقيمة واحد فإنه كلما قمت بإستدعاء الدالة سينفذ السطر الثاني فقط ويقفز عن السطر الأول لأنه موجود في الذاكرة.

دعه الآن له وقت سنشرحه بالتفصيل .

* `Main` لاحظ أن أول حرف كبير .

* `(string[] args)` وهي هنا تعني أننا بإمكاننا أن نستخدم الوسائط ((الباراميترات))

فمثلاً لو أنشأنا مشروع لجمع عددين وقمنا بتسميته Sum طبعاً ستقول لي بعد تنفيذ المشروع نكتب الجمل اللازمة لكي يقوم بالحساب في ما بينهما وذلك بالطلب من المستخدم أن يدخل رقمين مباشراً من طريق جمل الإدخال . حسناً هنا بإمكانك قبل تنفيذ المشروع أن تدخل العددين وتقوم بالتعامل معهما فمثلاً نذهب إلى محرر الدوس ونقوم بكتابة الجملة التالي :

C:\> Sum.exe 152 965

فنستطيع مباشرة وبأول جملة في المشروع أن تعطيه الناتج .

في السطر رقم (16) `//` لاحظ وجود قوسين هنا وهما لجمل التعليقات ((لاحظ الفرق في السطر الخامس)) . أي أنها جمل لا معنى لها تضع التوضيح أو التعليق على الجمل في هذا السطر

ملاحظة :

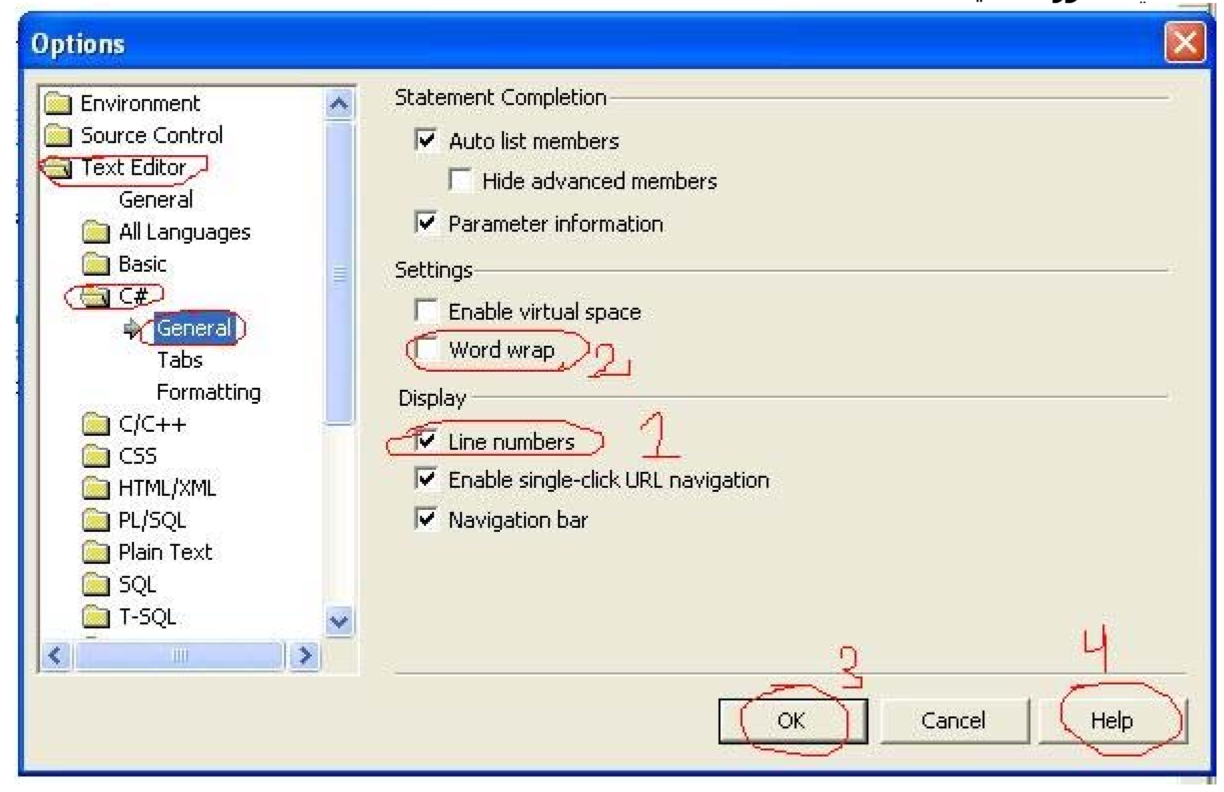
إذا واجهتك أي مشكلة في أي سطر وتريد معرفة المزيد قم بالنقر مرتين مزدوجتين على الجملة ثم قم بالضغط على F1 لظهور نافذة المساعدة بالجملة التي تريد فقط ((يجب أن تمتلك MSDN))

ملاحظة :

إذا لم تحتوي منطقة العمل على أرقام فبإمكانك إضافتها بإتباع المسار التالي :

Tools → Option → Text Editor → C# → General → Line Numbers

كما في الصورة التالية :



قم بوضع ✓ على الرقم ((1))

كذلك بإمكانك أن تفعل خاصية Word Wrap ((2)) والتي تعني أنه بعد حد معين للسطر قم بإنزاله إلى السطر الجديد وبذلك أنت بغنى عن شريط الـ Scroll Bar الأفقي .
أيضاً تستطيع إظهار معلومات أخرى عن هذه النافذة بالضغط على الرقم ((3)) .
الآن إختر كما في النافذة السابقة و إضغط الزر رقم ((3)) OK .

نكون هنا قد شرحنا كود البداية لملف السي شارب بطريقة الـ Console .

الآن سنقوم بكتابة أول برنامج شهير وهو طباعة جملة " Hello World " بواسطة المكتبة System :

الآن قم بكتابة السطر التالي في الدالة الرئيسية :

```
static void Main(string[] args)
{
    Console.WriteLine("Hello World");
}
```

ولاحظ أنه يوجد حروف كبيرة كما في الشكل ((تحته خط))
الآن للناقش السطر السابق :
*** إستخدمنا الدالة** `WriteLine` **والتي تسمح لنا بإظهار سطر على الشاشة وهي تأخذ قيمة من نوع String وبإمكانك أن تطبع المتغيرات والأسماء والأرقام .**
*** لاحظ أن الدالة السابقة خرجت من كلاس إسمه Console لتطبيق مبدأ الـ Full OO .**

يوجد أكثر من صيغة للدالة WriteLine . إليك الأمثلة كالتالي :

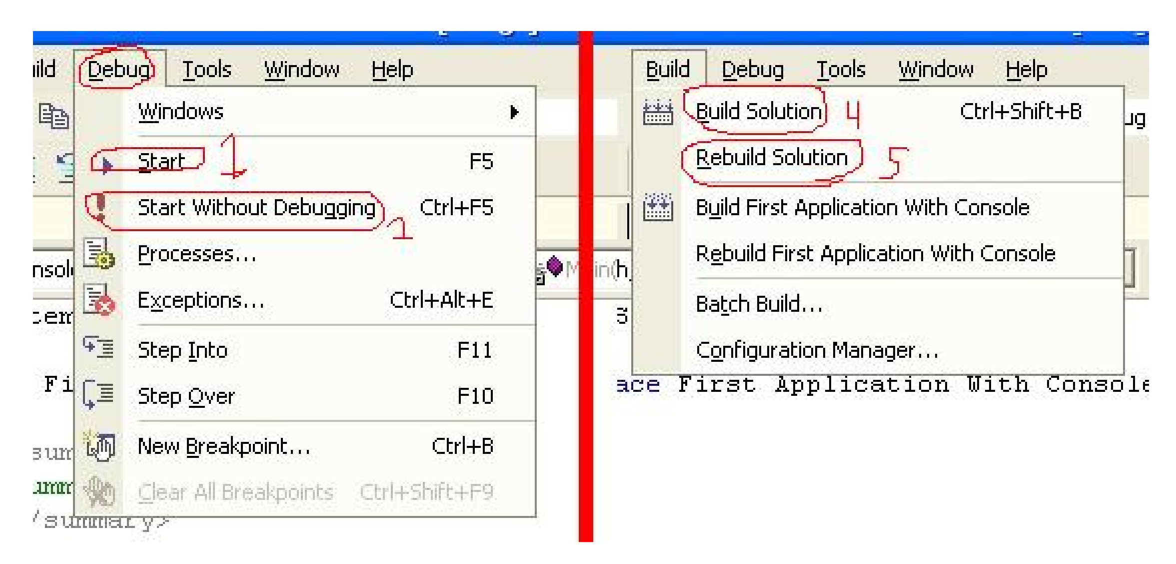
```
1 - Console.WriteLine("555");
2 - Console.WriteLine(555);
3 - Console.WriteLine("Hello To RTAQ");
4 - Console.WriteLine("Hello To {0}","RTAQ");
5 - Console.WriteLine("Sum {0} + {1} = {2}",5,4,5+4);
```

في المثال الأول :
سيكون الناتج 555 وسيعتبرها كأنها نص .
في المثال الثاني :
سيكون الناتج 555 وسيعتبرها كأنها رقم .
في المثال الثالث :
سيكون الناتج Hello To RTAQ وسيعتبرها كأنها نص .
في المثال الرابع :
سيكون الناتج Hello To RTAQ وسيعتبرها كأنها نص ولكن إنتبه هنا فيوجد تكنيك جديد وهو أننا بإمكاننا فصل الكلام إلى مواقع Index تعتبرها البيئة بالترتيب بداية من صفر فهنا قمنا بإضافة متغير ((نص)) إلى الجملة الأصلية Hello To الذي يمتلك Index = 0 فقمنا باستدعائه بالرمز {0} والموجود بعد الجملة الأولى مباشرة بعد الفاصلة .
في المثال الخامس :
سيكون الناتج 9 = 4 + 5 هنا أيضاً قام بأخذ المواقع Index بالترتيب ووضعها في الجملة الأولى .

والآن إلى كيفية تنفيذ البرنامج :

فبعد كتابة الكود قم بالذهاب إلى **Debug → Start Without Debugging** أو قم بالضغط على الأزرار التالية : **Ctrl + F5**

لعلك تتساءل : لماذا هذه مع أنه يوجد أوامر كثيرة مثل الـ Debug والـ Start ؟؟
سأقوم بتوضيح الأمر لك . أنظر إلى الصورة التالية :



الرقم 1 : وهو مسئول مباشرة عن ظهور النافذة السوداء وتطبيق الكود .
الرقم 2 : كوظيفة ((1)) ولكن هنا يقوم بوضع سطر إضافي وهو سطر إيقافي لمشاهدة الكود ففي الكود السابق وبطريقة الرقم ((1)) تظهر الشاشة السوداء ولكن سرعان ما تنتهي .
الرقم 3 : يقوم بإنشاء الملفات اللازمة للمشروع بما فيها الملف التنفيذي EXE .
الرقم 4 : يقوم بحذف الملفات السابقة وإنشاء ملفات جديدة حسب التغيرات التي حدثت .

بسم الله الرحمن الرحيم

المحاضرة الرابعة :

أنواع البيانات وجملة الإدخال والإخراج – قسم الـ Console :

كما تعلمنا سابقاً في جملة الطباعة أنه يمكننا الطباعة حسب المواقع Indexes .

سنقوم اليوم بالتوسع في صيغة جملة الإخراج والتعرف على جملة الإدخال وكذلك معرفة أنواع البيانات التي تتيحها لنا لغة السي شارب . لنبدأ الآن :

يمكن تعريف نوع من المتغيرات بالطريقتين التاليتين :

```
Data Type    Var Name   ;
Data Type    Var Name = Value   ;
```

سنقوم الآن بتعريف الأنواع كالتالي :

ظهر مفهوم المتغيرات (Variables)جديداً في عالم البرمجة مما أحدث تغييراً عميقاً في هذا المجال ويمكن تعريف المتغير بأنه إسم يحمل قيمة قابلة للتغيير في وقت لاحق في المشروع . ويخضع لشروط تسمية ويحمل نوع معين . أما بالنسبة للشروط فهي :

1 – أن تبدأ بحرف .
2 – يجوز إستخدام الرمز _ في البداية أو النهاية .
3 – يجوز إستخدام الأرقام في جميع الخانات ما عدا أول خانة .
4 – أن لا تكون من الكلمات المحجوزة للبرنامج .
5 – لغة السي شارب لغة حساسة أي (Age, AGE, aGE , AgE) كلها متغيرات لا يشبه بعضها بعضاً .

- سؤال : أي الجمل التالية برمجياً صحيحة . ولماذا؟
(جاوب عليها وأرسل الجواب على الإيميل الخاص بي مع ذكر الإسم وضع عنوانها HW_C#_1)

```
int idnumber;
int transaction_number;
int __my_phone_number__;
float 4myfriend;
float its4me;
double VeRyStRaNgE;
float while;
float myCash;
int CaseNo;
int CASENO;
int caseno;
```

أما بالنسبة للأنواع فهي كالتالي :

- تسمح لغة السي شارب بتعريف المتغيرات التالية :

Boolean , character, float , integer , double , decimal , string , object , long , short , byte

الأنواع باللون الأحمر (تحته خط) هي أنواع جديدة في لغة السي شارب حيث لم تكن موجودة في لغة السي ++ ، وأود إعلامك أخي العزيز إلى ضرورة كتابة الأنواع كلها بحروف صغيرة وأنت تعرف لماذا .

حسناً سأقوم الآن بشرح الأنواع جميعها :

1 – يعرف المتغير البولياني Boolean بالكلمة المحجوزة bool ويحتوي إحدى القيمتين true أو false صحيح أو خاطئ وإليك أخي الكريم مثالاً على ذلك :

```
bool myStatus = true;
bool yourStatus = false;
```

وكما رأينا في المثال السابق يمكن وضع قيمة إفتراضية في جملة التعريف أو في جملة لاحقة .
ويجب التنبيه هنا بأنه لا يمكن إستخدام الأرقام 0 أو 1 كما في الإصدارات السابقة .

2 – يعرف المتغير العددي الصحيح Integer بالكلمة المحجوزة int و هذا النوع هو من أكثر الأنواع إنتشاراً و إستعمالاً إليك أمثلة هذا النوع :

```
int count;
int number_of_students = 30;
```

وننبه هنا أن المتغير الذي يحمل نوع integer يأخذ قيمة عدد صحيح أي أننا إذا أسندنا إليه قيمة مثل 5.3 أو 5.9 فإنه سوف يعتبرها 5 .

3 – يعرف المتغير العددي العشري Float بالكلمة المحجوزة float وهو يشبه النوع السابق ولكن يسمح بالفاصلة العشرية مثال 3.33333333 . ومن هنا يمكننا إشتقاق النوع Double وهو النوع العشري ولكن يسمح بطول 32 قبل وبعد الفاصلة أيضاً النوع Decimal مشابه له . ومثال على النوع float :

```
float owned = 0.0f;
float owed = 1234567.89f;
```

هنا يجب التنويه إلى أمر وهو أن لغة السي شارب تعتبر النوع الأصل هو الـ Double فإذا أردت أن تعرف متغير من هذا النوع ما عليك سوى كتابة جملة التعريف له ثم قم بإعطائه قيمة وأنهي الجملة بفاصلة منقوطة . ولكن ...
مع الأنواع الأخرى مثل الـ Float والـ Decimal يجب أن تزيد حرف بعد التعريف فإذا أردت أن تعرف متغير من النوع Float عليك بكتابته كالتالي :

```
float x = 32.23f
```

حيث أن الحرف f يدل على أن هذا الرقم ليس من النوع Double وإنما من النوع Float
أيضاً إذا أردت أن تعرف متغير من النوع Decimal عليك بكتابة التالي :

```
decimal x = 31.43m
```

حيث أن الحرف m يدل على أن هذا الرقم ليس من النوع Double وإنما من النوع Decimal وهنا الحرف ميم للتمييز بينه وبين حرف الفاء للنوع Float .

4 – يعرف المتغير الخاني (خانات) Character بالكلمة المحجوزة char (لفظها كار وليس شار) ومن خلال هذا النوع الذي يحمل خانة واحدة فقط تستطيع تعريف أي خانة أو أي رمز من رموز لوحة المفاتيح أو الرموز الأمريكية المشفرة ASCII وإليك بعض الأمثلة على هذا النوع :

```
char firstInitial = 'J';
char secondInitial = 'K';
```

ومن هنا أيضاً يمكن أن نعرف النوع الجديد وهو الـ String وهو عبارة عن نوع كامل يحتوي على جمل ونصوص كتابية .

هنا يجب التنويه إلى أمر وهو أن لغة السي شارب تميز بين النوع Char والنوع String بالفاصلة حيث تعطي النوع Char فاصلة واحدة (') والنوع String بفاصلة مزدوجة (") وكما قلت لك النوع الأول يأخذ خانة واحدة فقط .

5 - النوع الأخير وهو الـ Object وهو عبارة عن نوع يحمل جميع أي قيمة من القيم السابقة أي أنه يمكن أن يحتوي عدد صحيح أو خاني أو من نوع بولياني ... الخ . ها هو مثال ذلك :

```
object x=313.22222m;
object x=313.22222f;
object x = 'a'
object x = "Hello"
```




ونهايةً أريد أن أذكر أننا عندما نعرف متغير من نوع ما فإن أول قيمة له هي مكانه في الذاكرة فمثلاً لو أننا قمنا بتعريف متغير من نوع صحيح وعملنا له إخراج كالتالي :

```
static void Main(string[] args)
{
      int  x;
      Console.WriteLine(x);

}
```

فإننا سوف نحصل على Error أي خطأ في كتابة الكود . حيث أن لغة السي شارب لا تسمح بطباعة مكان موقع المتغير في الذاكرة لذلك يجب أن نعطي له قيمة بداية كما في المثال التالي :

```
static void Main(string[] args)
{
      int  x = 124;
      Console.WriteLine(x);

}
```

كذلك يمكنك تعريف المتغيرات بالأنواع بإستخدام مكتبة الـ System كالتالي :

```
System.Int16 x=121;  → 16 character
System.Int32 x=121;  → 32 character
System.Int64 x=121;  → 64 character
```

أو أي نوع من الأنواع السابقة مثل :

```
System.String x="RTAQ";
```

الآن لنتعرف أكثر على جمل الإدخال والإخراج :
تعلمنا طباعة المتغيرات بطريقة الـ Indexes فمثلا :

```
int x=432;
Console.WriteLine("The Value Of X Is : {0}",x);
```

سأذكر في هذه الجملة أنه لا يجوز طباعة متغيران فقط مثل :

```
Console.WriteLine(x,y);
```

فهذه الجملة خاطئة ويجب أن تبدأ بنص وتصحيح الجملة السابقة كالتالي :

```
Console.WriteLine("The Value Of X & Y Is : {0}  {1}",x,y);
```

سيقوم بطباعة قيمة x .

الآن يمكننا وبواسطة جملة الإخراج التحكم في صيغة المخرج فمثلاً يمكننا إخراج المتغير على شكل عملة أو عدد صحيح أو بفاصلة عشرية أو حتى بالـ Exponential كالتالي :

```
Console.WriteLine("The Value Of X : {0:E}",x);
```

لاحظ هنا أننا قمنا بزيادة {0:E} أي موقع المتغير ثم الصيغة المطلوبة وحرف الـ E هنا يعني Exponential وهو متغير رياضي معروف .

حسناً إليك سرداً بقائمة الصيغ وما تعني والمخرجات بجانبه ((يجب أن يكون الحرف كبير)) :
على إفتراض أن قيمة x تساوي 52.19f .

```
Console.WriteLine("The Value Of X : {0:C}",x); → $ 52.190
Console.WriteLine("The Value Of X : {0:D}",x); → Error
Console.WriteLine("The Value Of X : {0:E}",x); → 5.219000E+001
Console.WriteLine("The Value Of X : {0:F6}",x); → 52.190000
Console.WriteLine("The Value Of X : {0:G}",x); → 52.19
```




الشرح :
في الجملة الأولى إستخدمنا الحرف C لإضافة تأثير العملة أي علامة الدولار والحرف C مأخوذ من إسم العملة بالإنجليزي وهي Currency .
في الجملة الثانية إستخدمنا الحرف D ومعناها Decimal وهي لا تنفع إلا مع النوع Integer فإذا عرفنا متغير من نوع عدد صحيح ثم أسندنا له القيمة 25 ثم نفذنا الجملة التالية لكان الناتج :

```
Console.WriteLine("The Value Of X : {0:D6}",x); → 000025
```

أي أنه أخذ 6 خانات للرقم x بوضع أصفار إلى اليسار لتصبح بعدد الخانات التي أدخلتها .
في الجملة الثالثة إستخدمنا الحرف E لإضافة تأثير المتغير الرياضي Exponential .
في الجملة الرابعة إستخدمنا الحرف F للدلالة على عدد الفاصلة ((الرقم بعد الحرف مباشرة)) .
في الجملة الخامسة إستخدمنا الحرف G وهو مأخوذ من كلمة General ولا يقوم بالتغيير بشيء .

أيضاً يمكننا إستخدام المعاملات المنطقية مثل AND أو OR كالتالي :
AND : يمكننا إستخدامها بالرمز & ومعناها (و) كالتالي :

```
int x,y;
x = 4;
y = 7;

Console.WriteLine(x&y);
```

هنا سيقوم بإستخدام النظام الثنائي كالتالي :
الرقم 4 يكافئ 100 بالنظام الثنائي والرقم 7 يكافئ 111 بالنظام الثنائي . الآن يقوم بأخذ خانة خانة إبتداءً من اليمين إلى اليسار ويقوم بتطبيق المعامل AND عليها :

1 & 1 → 1
1 & 0 → 0
0 & 1 → 0
0 & 0 → 0

فيصبح الناتج في الرقمين السابقين (100) بالتالي يكون الناتج رقم 4 لأنه يقوم بإعادتها إلى النظام العشري . وكذلك بالنسبة للمعامل OR إلا أنه يستعمل التالي :

1 | 1 → 1
1 | 0 → 1
0 | 1 → 1
0 | 0 → 0

يمكنك الحصول على رمز الـ OR بالضغط على \ + Shift . وهنا الناتج 111 ويكافئ 7 بالعشري .

هنا نكون أنهينا جملة الإخراج لنبدأ بجملة الإدخال :
- في لغة السي شارب لا تُقرأ الكلمة المدخلة إلا String أي أن هذه اللغة تقوم بتحويل ناتج الإدخال إلى نص فقط وإذا أردنا نوع آخر نقوم بالتغيير حسب جمل محددة وإليك تبيان ذلك :

جملة الإدخال والصيغة العامة :
Var Name = Console.ReadLine();

لاحظ هنا أننا قمنا بإسناد المدخل من لوحة المفاتيح إلى المتغير الذي قبل إشارة المساواة ويجب أن يحمل المتغير النوع String وإلا فسيظهر معك خطأ في تنفيذ المشروع .

إليك الأمثلة :

```
string x;
x = Console.ReadLine();
Console.WriteLine("You Entered : {0}",x);
```

سيقوم بإسناد قيمة المدخل إلى المتغير x كقيمة String .



حسناً لعلك تتساءل الآن كيف أقوم بإدخال رقم ؟؟!!
سأقوم بالرد عليك و أقول لك يجب هنا أن تستخدم النوع أو دالة الــ Convert الموجودة في المكتبة System كالتالي :

يمكنك أن تحول قيمة نص إلى رقم بإستعمال النوع ثم أتبعه بنقطة ثم كلمة Parse :

Integer Var = Type.Parse(String Var) ;

وإليك مثال ذلك :

```
static void Main(string[] args)
{
      string x;
      int y;
      x = Console.ReadLine();
      y = int.Parse(x) ;
      Console.WriteLine("Square Number : {0}",y*y);
}
```

في المثال السابق قمنا بتحويل المتغير x من نوع String إلى المتغير y من نوع Integer .
أيضاً يمكنك فعل السابق بالدالة Convert كالتالي :

```
y = Convert.ToInt16(x) ;
```

وهذا الكائن ((كائن التحويل)) يحتوي على جميع الأنواع .

ويمكنك دمج الخطوتان السابقتان في واحدة كالتالي :

```
static void Main(string[] args)
{
      int x;
      x = Convert.ToInt16(Console.ReadLine());
      Console.WriteLine("Square Number : {0}",x*x);
}
```

وهنا يرجع الأمر للمبرمج ولطريقة التفكير لديه ولكفاءته . فمن الممكن أن يكتب شخص برنامج بعشرة سطور وآخر يكتبها بخمسة سطور و بطريقة جيدة و عملية .

الآن سأقوم بإعطائك بعض الأسئلة :
(جاوب عليها وأرسل الجواب على الإيميل الخاص بي مع ذكر الإسم وضع عنوانها HW_C#_2)

1 – أكتب برنامج لجمع عددين بإدخال العددين من خلال المستخدم بإستخدام جملة الإدخال .
2 – أكتب برنامج لإيجاد المعادلة التالية $X^2+0.5X+1$ بإعتبار قيمة X تساوي 2 .
3 – أي الجمل التالية برمجياً خطأ . ولماذا؟

```
-  int x;
   x = Console.ReadLine();
   Console.WriteLine("Square Number : {0}",x*x);
-  Console.WriteLine(5,4);
- Console.WriteLine("who are you {1} Or {2} ?","Ahmad","Husam");
- int x,y;
   x = 4;
   y = 7;
   x = 5;
- Console.WriteLine("First Litter Of 1 & 2 Is : {0} , {1}",O,T);
- int x;
   x = 4;
   Console.WriteLine("The Value Of X Is : {0}",X);
```

بسم الله الرحمن الرحيم

المحاضرة الخامسة :

الجمل الشرطية وحلقات التكرار – قسم الـ Console :

في هذا الدرس سنتعلم كيفية كتابة شروط وكيفية قبولها أو عدم قبولها . أيضاً سنتعلم كيف نستخدم الجمل التكرارية لتنفيذ جمل مرات معينة .

الآن لنبدأ في قسم جمل الإختيار حيث تقسم جمل الإختيار Selection Statement كالتالي :

If Statement .
Switch Statement .

الجملة الشرطية If :

يمكننا إستخدام الجملة الشرطية لوضع شروط وجواب لهذه الشروط
فمثلاً نعلم أننا لا نستطيع القسمة على صفر وإن نفذنا ذلك فسيظهر لنا خطأ في نافذة الـ Task list
لذلك لا بد من وضع شرط لعدم إمكانية إستخدام الصفر في المقام كالتالي :

```
if ( condition )
      Statement ;
```

ويتكون الشطر Condition من إسم المتغير والمعامل والقيمة كالمثال التالي :

```
int y;
y = int.Parse(Console.ReadLine());

if ( y == 0 )
      Console.WriteLine("Error");
```

هنا كان إسم المتغير Y والمعامل رمز المساواة (عندما نكون في مقارنة يجب وضع رمز المساواة مرتين) والقيمة 0 . فإذا قمنا بإدخال صفر كقيمة للمتغير Y فإنه سيظهر لنا الجملة Error كما وضعناها في جملة الشرط وهذه الحركة تعتبر جواباً للشرط

حسناً لو أننا أردنا أن نضع البديل للشرط أي أنه إذا لم تنفذ الجملة ماذا يحدث .
نستطيع فعل ذلك بإستخدام جملة else وهي مرادفة للجملة الشرطية .
فيصبح المثال السابق كالتالي :

```
int x,y;
x = int.Parse(Console.ReadLine());
y = int.Parse(Console.ReadLine());

if ( y == 0 )
      Console.WriteLine("Error");
else
      Console.WriteLine("X / Y = {0}",x/y);
```

ونستطيع أيضاً إستخدام الجملة Else if إذا كنا نريد تنفيذ شرط واحد فقط بمعنى أن المترجم يقوم بالمرور على جملة واحدة فقط و يهمل الباقي .

```
int x;
x = int.Parse(Console.ReadLine());

if ( x == 1 )
      Console.WriteLine("A");
else if ( x == 2 )
      Console.WriteLine("B");
else if ( x == 3 )
      Console.WriteLine("C");
else
      Console.WriteLine("Other");
```

لذلك في المثال السابق ستنفذ جملة واحدة فقط .

سـنناقش الآن مثالاً لنتعرف بعض الدوال التي تتيحها لنا لغة السـي شارب :

```
static void Main(string[] args)
{
      Console.Write("Enter Any Character : ");
      char c = char.Parse(Console.ReadLine());
      if ( char.IsUpper(c))
        Console.WriteLine("The Character Is Upper Case");
      else if ( char.IsLower(c))
        Console.WriteLine("The Character Is Lower Case");
      else if ( char.IsDigit(c))
        Console.WriteLine("The Character Is Number");
      else
        Console.WriteLine("The Character Is Not Alphanumeric");
}
```

في هذا المثال يقوم المستخدم بإدخال أي خانة من لوحـة المفـاتيح ويقـوم البرنـامج بتحديـد هـل هـو حرف صغير أو كبير أو رقم أو غير ذلك .

لاحظ أننا هنا إستخدمنا بعض الدوال للكائن Char وهـي عبـارة عـن دوال تعيـد إحـدى القيمتـين : إمـا True أو False وذلك بتحديد بعض الأمور المتعلقة بالخانات .

ولاحظ أيضاً أن البرنامج لا يدخل إلا لجملة واحدة فقط من الجمل الشرطية السـابقة .

جملة الإختيار Switch :

هنا الأمر مشابه لجملة الشـرط IF وسـبب وجود هذه الجملة هو أنه في حالة إذا أردنا أن نختـار أو أن نبني عدة شروط كما في جملة الـ If فقـط بـدون إسـتخدام جملـة الــ if else هـذا لـن يكـون مفيـد لنـا فسوف يعمل البرنامج على المرور على كل جمل الشـرط وسـيزيد الوقـت للتنفيـذ ومـن هـذا الـسبب خرجت هذه الجملة .

لنأتي إلى الصيغة العامة لجملة Switch :

```
switch ( var )
      case value :
            statement;
      break ;
```

وإليك عزيزي المثال على كيفية إستخدامها :

```
static void Main(string[] args)
{
      Console.Write("Please Enter Your Selection 1,2 Or 3 : ");
      int x = int.Parse(Console.ReadLine());

      switch ( x )
      {
            case 1 :
                  Console.WriteLine("Your Choose 1 .");
                  break ;
            case 2 :
                  Console.WriteLine("Your Choose 2 .");
                  break ;
            case 3 :
                  Console.WriteLine("Your Choose 3 .");
                  break ;
            default :
                  Console.WriteLine("Error:?: Please Choose 1 Or 2 Or 3 .");
            break;
      }
}
```

في هذا المثال طلبنا من المستخدم تحديد خيار ضمن الأعداد 1 و 2 و 3 و عندما وصل إلى جملة الإختيار Switch قام بالذهاب فوراً إلى الخيار المطلوب .
نكون هنا قد إنتهينا من جمل الإختيار Selection Statement ولنبدأ في جمل تالية .
الآن سنناقش الجمل الحدّية وحلقات التكرار Iteration Statement كالتالي :

Do _ While Loop Statement
While Loop Statement
For Loop Statement

جملة التكرار Do :
تعتبر هذه الجملة من جمل التكرار لشرط معين فمثلاً إذا أردنا أن نجري جملاً حتى تنفيذ شرط معين فإننا نستخدم هذه الجملة ويمكن كتابة الصيغة العامة لهذه الجملة كالتالي :

```
do
{
      Statement ;
}while ( Condition ) ;
```

فلاحظ معي هنا أن البرنامج يقوم أولاً بتنفيذ الجمل الموجودة داخل الحلقة ثم يقوم بإختبار الشرط يعني سينفذ الجمل الموجودة داخل الحلقة مرة واحدة فقط على الأقل.
وإليك مثال على هذه الجملة :

```
static void Main(string[] args)
{
      int x = 5;
      do
      {
            Console.WriteLine("The Value Of X Is : {0}",x);
            x = x + 1 ;
      }while ( x < 10 ) ;
}
```

في هذا المثال سيقوم أولاً بطباعة الرقم 5 ثم يزيد قيمة x ثم يتأكد من تنفيذ الشرط أو عدمه وطبعاً قيمة 5 أصغر من قيمة 10 لذلك سوف يستمر في طباعة قيمة المتغير x حتى يصل إلى قيمة 9 فيقوم بطباعة قيمة x والتي تساوي 9 ثم يزيد قيمة المتغير ليصبح 10 بعدها سيقوم بالتأكد من قيمة x فنقارن هنا هل الرقم 10 أقل من الرقم 10 وطبعاً هذا خطأ لذلك سوف يخرج من الجملة التكرارية .

جملة التكرار For :
من أشهر جمل التكرار وهي هنا تقوم بالزيادة التلقائية مع تطبيق الشرط أولاً .
والصيغة العامة لهذه الجملة هي :

```
for ( var = initial value ; Condition ; Increasing / decreasing)
{
      Statement ;
}
```

كما قلت لك سابقاً فهنا يقوم البرنامج بالتأكد من الشرط ثم يعمل على تنفيذ الجمل داخل الحلقة .
و هاك المثال التالي :

```
static void Main(string[] args)
{
      for (int i = 1 ; i < 10 ; i++ )
      {
            Console.WriteLine("Value Of i Is : {0}",i);
      }
}
```

سيقوم بتنفيذ الجمل وطباعة المتغير i من قيمة 1 إلى قيمة 9 .
لعلك تتساءل ما هو تعبير ((i++)) ؟
والجواب أن هذا التعبير كناية عن زيادة المتغير i بقيمة 1 ويمكنك فهمها بالجملة التالية :

```
i = i + 1 ;
```

ويمكنك إستخدام تعبير الإنقاص بالجملة التالية : --i وكذلك يمكنك فهما كالتالي :

```
i = i - 1 ;
```

الآن إستخدم تعبير ++i و سنأخذ تفسيراً لهذا التعبير في الدروس القادمة .

جملة التكرار While :

تشبه إلى حد كبير في مبدأها عمل جملة التكرار Do ولنتحدث الآن عن الصيغة العامة لهذه الجملة :

```
while ( Condition )
{
      Statement ;
}
```

هنا سيبقى يحقق الشرط وإذا خالف الشرط يقوم بالخروج من هذه الحلقة .
وإليك مثال على هذه الجملة :

```
static void Main(string[] args)
{
      int x = 1 ;
      while ( x != 5 )
      {
            Console.WriteLine("X = {0}",x);
            x++ ;
      }
}
```

و أنبه هنا :
بالنسبة للجمل التكرارية يمكن أن نعرّف مفهوم المالا نهاية في أنه إذا إستخدمت أحد جمل التكرار ولم تتمكن من تحقيق شرط الخروج . وهنا سيقوم البرنامج بالتنفيذ إلى مالا نهاية ولن يخرج من البرنامج أبداً لذلك إحذر في إستخدام هذه الجمل وأيضاً في جملة الشرط .
كذلك في الأمثلة السابقة إذا قمت بتغيير مكان جملة الطباعة بدلاً من مكان جملة الزيادة سيقوم بالتغيير الكلي للجملة .

وبهذا نكون قد غطينا جمل الإختيار والتكرار .

الآن سأقوم بإعطائك بعض الأسئلة :
(جاوب عليها وأرسل الجواب على الإيميل الخاص بي مع ذكر الإسم وضع عنوانها HW_C#_3)

1 – أكتب برنامج لطباعة الأعداد من 1 إلى 100 كل خمس أرقام كالتالي : 5,10,15,20,…..,90,95,100
وذلك بإستخدام جملة التكرار While .
2 – ارسم الشكل التالي بواسطة جملة التكرار For :

```
*****
****
***
**
*
```

3 – بدون كتابة البرنامج وتنفيذه على الحاسوب ما مخرجات الكود التالي :

```
static void Main(string[] args)
{
      Console.WriteLine("X    1     2     3     4     5");
      Console.WriteLine("  *---------------------------------------");
      for (int i = 1 ; i <= 5 ; i++ )
      {
            Console.Write("{0} |    ",i);
            for (int j = 1 ; j <= 5 ; j++ )
            {
                  Console.Write("{0}      ",i*j);
            }
            Console.WriteLine("");
      }
      Console.WriteLine("  *---------------------------------------");
}
```




بسم الله الرحمن الرحيم

المحاضرة السادسة :

المصفوفات والحلقات المصاحبة لها ومعاملات الزيادة والنقصان – قسم الـ Console :

المصفوفات Array :

تتيح لنا لغة السي شارب التعامل مع المصفوفات كالتالي :

```
Type []name = new Type[Size];
```

وهنا نذكر نوع المصفوفة ثم نكتب رمز المصفوفة قبل إسم المصفوفة ونتبعهما بعلامة المساواة ونكتب الكلمة المحجوزة new ثم نفس النوع الذي كتبناه أول الجملة ثم نذكر الحجم وإليك مثال على ذلك :

```
int []x = new int[3];
```

و بإستخدام الكلمة المحجوزة new تتقدم مفهوم المصفوفة للتعامل معها على شكل مؤشرات Pointer وذلك للخروج عن المألوف كونها Dynamic بدلاً من Static و بهذه الخطوة أصبحت لغة السي شارب تشبه الجافا في عملها .

في الجملة السابقة قمنا بحجز مكان لها بالذاكرة ولم نعطها قيم أولية ، لذلك يمكننا إعطائها قيم أولية بطريقتين مختلفتين وهما :

1 – في أثناء حجز المصفوفة يمكننا إعطائها قيم أولية كما في المثال التالي :

```
int []x = new int[5]{4,3,7,22,8};
```

فهنا قمنا بإسناد قيم للمصفوفة فوراً وفي سطر واحد .

2 – بعد التعريف للمصفوفة يمكنك إعطائها القيم كما في المثال التالي :

```
int []x = new int[3];
x[0] = 21 ;
x[1] = 63 ;
x[2] = 7 ;
```

كذلك يمكنك إعطاء المصفوفة أكثر من بُعد كالتالي :

```
int [,]x = new int[2,3];
```

فهنا قمنا بوضع فاصلة للدلالة على أنه يوجد لدينا هنا مصفوفة ذات بُعدين .

كذلك يمكنك إعطاء القيم الأولية لها بطريقتين :

1 - في أثناء حجز المصفوفة يمكننا إعطائها قيم أولية كما في المثال التالي :

```
int [,]x = new int[2,3]{{1,4,2},{52,12,9}};
```

2 – بعد التعريف للمصفوفة يمكنك إعطائها القيم كما في المثال التالي :

```
int [,]x = new int[2,3];
x[0,0] = 14;
```

وهنا يجب أن نوضح بعض التنبيهات ومن ضمنها أن لغة السي شارب تعتبر صفرية البداية Zero Base يعني أول عنصر في المصفوفة يحمل الرقم 0 وليس 1 كما يتبادر إلى الأذهان .

بمعنى أنه إذا أردت الحصول على أول عنصر يجب عليك فعل التالي :

```
int []x = new int[2]{1,3};
Console.WriteLine(x[0]);
```

وأيضا القيم الأولية للعناصر التي لم تعطها قيم هي 0 في النوع الصحيح وحسب كل نوع ففي النوع الخانئي تكون القيمة الأولية هي خانة الفراغ Null وهكذا .



ويمكنك التعامل مع المصفوفة كما تريد فهنا سنقوم بكتابة برنامج يطلب من المستخدم إعطاء القيم الأولية للمصفوفة ومن ثم طباعتها كالتالي :

```
static void Main(string[] args)
{
      int []x = new int[5];
      for (int i=0 ; i<5 ;i++)
      {
            Console.Write("Enter Value At Location {0} In Array : ",i+1);
            x[i]=int.Parse(Console.ReadLine());
      }

      Console.WriteLine();
      for (int j=0 ; j<5 ;j++)
            Console.WriteLine("Value Of Location {0} Is : {1}",j+1,x[j]);
}
```

وكذلك يمكنك التعامل مع المصفوفة ذات الأبعاد المختلفة .

وكذلك يمكنك معرفة حجم المصفوفة بالكلمة المحجوزة Length كما في الجملة التالي :

```
Console.WriteLine(x.Length);
```

والآن سأتحدث عن حلقة تكرارية جديدة صدرت في لغة السي شارب وهي خصوصية من خصوصيات المصفوفات وهي جملة الـ foreach وإليك الصيغة العامة لها :

```
foreach ( Type var in Array )
```

ووظيفة هذه الجملة هي المرور على كل عنصر من عناصر المصفوفة وأخذ القيمة الموجودة فيه ووضعها في المتغير Var وإليك المثال التالي على كيفية كتابة الحلقة:

```
int []x = new int[5];
foreach ( int y in x )
```

وهاك مثال على كيفية حساب مجموع عناصر المصفوفة :

```
static void Main(string[] args)
{
      int []x = new int[5]{12,41,8,3,6};
      int sum = 0 ;
      foreach ( int y in x )
            sum+=y;
      Console.WriteLine(sum);
}
```

ومن أهم الخصائص لهذه الحلقة هي أنه إذا كانت عندك مصفوفة ذات أبعاد كبيرة مثلاً 5 فإنك غير مضطر لكتابة خمس حلقات For . وكذلك تريحك من كتابة حجم المصفوفة في كل حلقة .

وسأتطرق للنوعين الموجودين في مثل هذه الحلقات :

```
foreach ( Type var in Array )
```

فعندك هنا النوع للمتغير وهو باللون الأحمر
والنوع للمصفوفة وهو باللون الأخضر

وفي كل الأنواع سيفهم النوع باللون الأحمر النوع باللون الأخضر حسب فهمه فإذا كان النوع الأحمر عدد صحيح وكان اللون الأخضر خانات فسيقوم بتحويل الخانة حسب رقمها بالآسكي كود ومن ثم يقوم بالعمليات عليها وهكذا لجميع الأنواع .

معاملات الزيادة والنقصان :

تتيح لك لغة السي شارب زيادة المتغيرات من النوع الصحيح بعدة طرق فمن أشهرها وهي الطريقة التقليدية كتابة المتغير في جهة والزيادة عليه في جهة أخرى :

```
Var = Var (operation) value ;
```




فيمكنك بدلاً من كلمة Operation كتابة أي عملية مثل + * - / .
وإليك مثالاً على ذلك والذي يقوم بزيادة المتغير بقيمة 1 :

```
i = i + 1 ;
```

والأغلب يستعمل هذه الطريقة .
وأيضاً يمكنك إستعمال جملة بدلاً من كتابة المتغير مرتين كالتالي :

```
Var (operation)= value ;
```

كما في المثال التالي :

```
i += 4 ;
```

فهنا نقوم بزيادة المتغير بقيمة 4 .
والآن سنتحدث عن متغيرات الزيادة بقيمة واحد فقط فهنا يمكنك زيادة متغير بقيمة 1 فقط بالتعبير :

```
Var++ ;
```

و أيضاً يمكنك إستعمال التعبير :

```
++Var ;
```

وأيضاً يمكنك التعبير بالنقصان كما يلي :

```
Var-- ;
--Var ;
```

حسناً سأقوم بالتعريف عن الصيغتان وقول وظيفة كل واحد منهما :

أولاً التعبير ++i ، --i :
هنا يقوم المترجم بزيادة المتغير بعد المرور على السطر و الإنتقال إلى السطر التالي :
إليك مثالاً على ذلك :

```
static void Main(string[] args)
{
	int i=5 ;
	Console.WriteLine(i++);
	Console.WriteLine(i);
}
```

هنا سيقوم بطباعة قيمة 5 ثم بالإنتقال إلى السطر التالي سيقوم بزيادة المتغير .
وأنا أتحدث هنا عن معدل الزيادة وكذلك الحال بالنسبة للنقصان .

أولاً التعبير ++i ، --i :
هنا يقوم المترجم بزيادة المتغير في نفس السطر و الإنتقال إلى السطر التالي :
إليك مثالاً على ذلك :

```
static void Main(string[] args)
{
	int i=5 ;
	Console.WriteLine(++i);
	Console.WriteLine(i);
}
```

هنا سيقوم بزيادة المتغير و سيقوم بطباعة قيمة 6 ثم بالإنتقال إلى السطر التالي وطباعة 6 في الجملة التالية .

جملة Break وجملة Continue في حلقات التكرار :

تتيح لنا لغة السي شارب عوامل الهروب من الحلقات فبإمكاننا الخروج من حلقة معينة إذا حدث شرط معين وكذلك يمكننا تخطي حلقة معينة عند حدوث حدث نقوم بتحديده .
جملة Break :
وهنا تكتب كلمة break صريحة داخل أي حلقة من حلقات التكرار وهي غالباً تكون ضمن جملة شرطية معينة كما في المثال التالي :

```
static void Main(string[] args)
{
	for ( int i=0 ; i<10 ; i=i+2)
	{
		if ( i == 6 ) break ;
		Console.WriteLine(i);
	}
}
```

في المثال السابق سيقوم البرنامج بزيادة المتغير بقيمة 2 فستكون أول قيمة له هي 0 ثم 2 ثم 4 ثم عندما يصل إلى القيمة 6 سيقوم بالإستفسار عن قيمة المتغير هل قيمته 6 ؟ فتتحقق الجملة وتقوم بعمل Break والخروج من الحلقة نهائياً .

جملة Continue :

وهنا تكتب كلمة Continue صريحة داخل أي حلقة من حلقات التكرار وهي غالباً تكون ضمن جملة شرطية معينة كما في جملة Break وإليك مثال ذلك :

```
static void Main(string[] args)
{
      for ( int j=0 ; j<=5 ; j++ )
      {
            if ( j == 3 )
                  continue ;
            Console.WriteLine(j);
      }
}
```

في المثال السابق سيقوم البرنامج بزيادة المتغير بقيمة 1 فستكون أول قيمة له هي 0 ثم 1 ثم 2 ثم عندما يصل إلى القيمة 3 سيقوم بالإستفسار عن قيمة المتغير هل قيمته 3 ؟ فتتحقق الجملة وتقوم بعمل Continue فسيقوم بالتخطي والهروب من الحلقة وإكمالها في وقت لاحق .

الآن سأقوم بإعطائك بعض الأسئلة :
(جاوب عليها وأرسل الجواب على الإيميل الخاص بي مع ذكر الإسم وضع عنوانها HW_C#_4)

1 - قم بإنشاء مصفوفة وخزن فيها مربع الموقع الخاص بها وضح حجمها 500 فمثلاً الموقع رقم 0 سيحمل قيمة في موقعه وهي 0 والموقع رقم 1 سيحمل قيمة 1 والموقع 2 سيحمل القيمة 4 وهكذا.

2 – قم بملء عناصر مصفوفة تحمل الحجم 55 بحيث تقوم بتعبئة إشارة النجمة (*) في كل عنصر من مضاعفات الرقم 6 ولا تمر على العناصر التي تحمل رقم عناصر من معاملات الرقم 10 ، وقم بتعبئة الباقي بالحرف (O) . فمثلاً الموقع 0 لا ينطبق عليه أي شرط إذن نقوم بتعبئته بالخانة (O) والعنصر السادس تنطبق عليه إشارة النجمة فنقوم بتعبئة بإشارة النجمة والموقع 10 نتركه فارغ وهكذا . وإذا كتبته بشكل صحيح سيخرج معك الناتج كالتالي :

Value Of 0 Is : *	Value Of 20 Is :	Value Of 40 Is :
Value Of 1 Is : O	Value Of 21 Is : O	Value Of 41 Is : O
Value Of 2 Is : O	Value Of 22 Is : O	Value Of 42 Is : *
Value Of 3 Is : O	Value Of 23 Is : O	Value Of 43 Is : O
Value Of 4 Is : O	Value Of 24 Is : *	Value Of 44 Is : O
Value Of 5 Is : O	Value Of 25 Is : O	Value Of 45 Is : O
Value Of 6 Is : *	Value Of 26 Is : O	Value Of 46 Is : O
Value Of 7 Is : O	Value Of 27 Is : O	Value Of 47 Is : O
Value Of 8 Is : O	Value Of 28 Is : O	Value Of 48 Is : *
Value Of 9 Is : O	Value Of 29 Is : O	Value Of 49 Is : O
Value Of 10 Is :	Value Of 30 Is : *	Value Of 50 Is :
Value Of 11 Is : O	Value Of 31 Is : O	Value Of 51 Is : O
Value Of 12 Is : *	Value Of 32 Is : O	Value Of 52 Is : O
Value Of 13 Is : O	Value Of 33 Is : O	Value Of 53 Is : O
Value Of 14 Is : O	Value Of 34 Is : O	Value Of 54 Is : *
Value Of 15 Is : O	Value Of 35 Is : O	Value Of 40 Is :
Value Of 16 Is : O	Value Of 36 Is : *	Value Of 41 Is : O
Value Of 17 Is : O	Value Of 37 Is : O	Value Of 42 Is : *
Value Of 18 Is : *	Value Of 38 Is : O	Value Of 43 Is : O
Value Of 19 Is : O	Value Of 39 Is : O	Value Of 44 Is : O

3 – قم بحساب المخرجات في البرنامج التالي بدون إستخدم الحاسوب((لا تستعمل الحاسوب)) :

```
static void Main(string[] args)
{
      int x = 6 ;
      int y = 2 ;
      int z = 8 ;

      x += 2*y;
      y++;
      z = ++x+y--+ 5 ;
      x= ++x+ ++y ;
      Console.WriteLine("The Value Of x Is : {0}",x);
      Console.WriteLine("The Value Of y Is : {0}",y);
      Console.WriteLine("The Value Of z Is : {0}",z);
}
```

4 – ما مخرجات البرنامج التالي وبدون إستخدام الحاسوب :

```
static void Main(string[] args)
{
      for ( int i=0 ; i<5 ; i++)
      {
            for ( int j=0 ; j<3 ; j++)
            {
                  Console.Write("O");
                  for ( int k=0 ; k<3 ; k++ )
                  {
                        Console.Write("M");
                        if ( k+j%2 == 0 )
                        {
                              Console.Write("S");
                              continue;
                        }
                  }
            }
            Console.WriteLine("");
      }
}
```

5 – اكتب برنامج لخزن القيمة (الموقع ضرب 5) يعني الـ 5 * Index في مصفوفة حجمها 20 .

بسم الله الرحمن الرحيم

المحاضرة السابعة :

برمجة الأنواع وعملية الـ Casting وإستخدام جملة الـ Goto وتقسيم البرنامج – قسم الـ Console :

برمجة الأنواع وتصميمها بواسطة الكلمة المحجوزة enum :

تتيح لنا لغة السي شارب صناعة أنواع جديدة غير المعروفة وإعطاء نطاق لها فمثلاً لو أنك مضطر لأن تستخدم متغير من نوع الأسبوع يعني أنك تريد إعطاء المتغير من هذا النوع قيمة أحد الأيام الموجودة في الأسبوع وقمت بالبحث عن نوع لفعل ذلك فلن تجده . لذلك سهلت علينا لغة السي شارب وقامت بمنحنا كلمة Enum والتي تقوم بعمل نوع جديد وإليك الصيغة العامة لها :

```
enum Type { Val1 , Val2 , ... , Val* } ;
```

وهاك مثال على كيفية إستعمالها لصناعة نوع الأسبوع :

```
enum week { Sat , Sun , Mon , Tue , Wen , Thu , Fri } ;
```

وأنبه هنا أنه يجب عليك كتابتها قبل الـ Class يعني ليس داخل الدالة الرئيسية وإنما داخل الكلاس الحامل للدالة الرئيسية أو في منطقة بين كلمة namespace وكلمة class .
وإليك مثال على ذلك :

```
using System;

namespace First_Application_With_Console
{
      enum week { Sat , Sun , Mon , Tue , Wen , Thu , Fri } ;
      class Class1
      {
            [STAThread]
            static void Main(string[] args)
            {
                  week w1 = week.Tue ;
                  Console.WriteLine(w1);
            }
      }
}
```

في المثال السابق قمنا بإنشاء نوع جديد وقمنا بإنشاء كائن منه وإعطائه قيمة من القيم المتاحة له . ولعلك تسأل لماذا لا نقوم بإعطائه القيمة مباشرةً لماذا يجب علينا كتابة إسم النوع ؟ والجواب أن لغة السي شارب لا تعتبر هذا النوع من الأنواع الموجودة أصلاً في اللغة يعني ليست كلمة محجوزة لذلك يجب علينا كتابة إسم النوع الجديد ثم إتباعه بقيمته الجديدة .

تحويل الأنواع بواسطة الـ Casting :

إذا عرفنا متغيرين من نوع int والآخر من نوع float وقمنا بإسناد قيمة الـ float إلى المتغير الصحيح سيقوم البرنامج بإرسال رسالة Error ويقول لك الخطأ أنه لا يمكنك إسناد القيمة .
كما في المثال التالي :

```
static void Main(string[] args)
{
      int a = 3 ;
      float b = 6.12f ;
      a = b ;
      Console.WriteLine(a);
}
```

هنا تتيح لنا لغة السي شارب إنشاء تحويل أي قيمة إلى قيمة من نوع آخر . فكما في الحالة السابقة يمكنك طباعة المتغير بإسناد القيمة الجديدة له ولكن النوع القديم سيأخذ القيمة الجديدة كما يفهمه وبلغته الخاصة فمثلاً كما في المثال السابق وعند إسناد القيمة الجديدة سيأخذها بدون كسور عشرية ويأخذ فقط الجزء الصحيح .

وإليك تصحيح للمثال السابق بإستخدام عملية الـ Casting :

```
static void Main(string[] args)
{
	int a = 3 ;
	float b = 6.12f ;
	a = (int)b ;
	Console.WriteLine(a);
}
```

لاحظ هنا أننا إستخدمنا النوع الذي نريد تحويل القيمة الجديدة لديه وهذه ميزة من ميزات لغة السي شارب عن اللغات السابقة وقد أخذت الفكرة من لغة الجافا .

نعود الآن لصياغة الأنواع الجديدة بإستخدام كلمة enum فهنا يمكننا أيضاً إستخدام الـ Casting معه أيضاً . فمثلاً إذا كتبنا المثال التالي :

```
using System;
namespace First_Application_With_Console
{
	enum week { Sat , Sun , Mon , Tue , Wen , Thu , Fri } ;
	class Class1
	{
		[STAThread]
		static void Main(string[] args)
		{
			week w1 = week.Tue ;
			int a = 2 ;
			w1 = (week)a ;
			Console.WriteLine(w1);
		}
	}
}
```

هل تستطيع معرفة الناتج ؟
في المثال السابق قام المترجم بإعطاء قيمة المتغير w1 الرقم 2 ولو قلنا لأنفسنا كيف سيفهم النوع week الرقم 2 وما طريقته في ذلك ؟
حسناً سيفهمها المترجم كالتالي :
سيقوم بإعطاء القيم الموجودة في النوع الجديد بالترقيم إبتداءاً من الصفر إلى آخر قيمة . ولاحظ معي لو رقمناها ما هي القيمة التي تحمل رقم 2 ؟
أكيد ستعرف الجواب وتقول لي هي القيمة Mon وسأقول لك صحيح .

حسناً ماذا لو قلت لي أنا حر في ترقيم القيم وأريدها قيماً عشوائية فكيف أفعل ذلك ؟
يمكنك فعل ذلك ببساطة و عند تعريف النوع الجديد يمكنك إعطائها القيم بالعدد الصحيح كالتالي :

```
enum week { Sat=15 , Sun=4 , Mon=162 , Tue=55 , Wen=0 , Thu=76 , Fri=11 } ;
```

وذلك بإضافة رمز المساواة وإعطائه قيمة كما تريد .
وإليك مثال ذلك :

```
using System;
namespace First_Application_With_Console
{
	enum week { Sat=15 , Sun=4 , Mon=162 , Tue=55 , Wen=0 , Thu=76 ,
			Fri=11 } ;
	class Class1
	{
		[STAThread]
		static void Main(string[] args)
		{
			week w1 = week.Tue ;
			int a = 55 ;
			w1 = (week)a ;
			Console.WriteLine(w1);
		}
	}
}
```

هنا قمت بتلبية طلبك ...

جملة القفز غير التسلسلي GoTo :

ماذا لو أنك وصلت إلى جملة و اضطررت للرجوع إلى سطر معين كيف يمكنك فعل ذلك ؟
تتيح لك سي شارب وككل اللغات جملة goto والموجودة في معظم اللغات وهي مسئولة عن القفز إلى سطر معين ، ويجب إستعمال ليبل Label معها كالتالي :
نقوم بكتابة سطر كامل ويحتوي على كلمة تتبعها علامة النقطتان الرأسيتان (:) وأنت حر في إستخدام إسم الليبل ولكن لا تختار كلمة محجوزة .
ثم في قسم ثاني وغالباً يكون جملة شرطية وتستدعيها كالتالي :

```
goto Label ;
```

ولكي تستوعب هذه الجملة إليك المثال التالي :

```
static void Main(string[] args)
{
	int x=0;
	Label:
	Console.WriteLine(++x);
	if (x != 5 )
		goto Label ;
}
```

ماذا تلاحظ هنا ؟
فعلت جملة goto كفعل حلقة من حلقات التكرار ..
لها إستخدامات كثيرة ومفيدة ولكن هذه الجملة تعتبر ضد مبدأ الـ OO ولكن لم تتمكن أي لغة من حذفها لأن لها فوائد وتاريخ قديم .
ومن أكثر الفوائد أنها تعمل على تشتيت المؤشر Cursor بدلاً من العمل المتسلسل Sequential وتمنحنا حق التوجه إلى أي مكان في البرنامج من خلال الليبل المصاحب لها .
ونظم التشغيل تستفيد منها فمثلاً لعمل إسترجاع للجهاز Recovery في حالة حدوث خطأ معين في نفس النظام .

كفاءة البرنامج والتقسيمات من خلال الدوال Functions :
من أهم خصائص البرمجة ومن أهم مزاياها أنها تتيح لك تقسيم البرنامج إلى دوال صغيرة لتسهيل البرنامج وللكفاءة وأيضاً لإكتشاف الأخطاء بوقت أقل .

فهنا لغة السي شارب تتيح لك كتابة دوال في الكلاس الخاصة بك بالشكل التالي :

```
Type Function_Name ( Type Var , ... )
{
	Statement ;
	.... ;
	return Ret_value ;
}
```

بداية نشرح نوع الدالة وهي أول كلمة هنا وهنا يمكنك إعطائها الأنواع المعروفة مثل int و أيضاً char وأي نوع من الأنواع الموجودة عندك . وهذا النوع يؤثر بشكل مباشر على القيمة المسترجعة من الدالة بمعنى أنه إذا كان نوع الدالة عدد صحيح يجب أن نرجع في المتغير `Ret_value` **قيمة من النوع العددي الصحيح .**
وأما الجمل الموجودة داخل الدالة فهي الجمل التي تؤديها هذه الدالة والوظيفة التي كتبت لها .
وأما المتغيرات والتي داخل القوسين ( ) فهي تسمى الوسائط Parameter وهنا يمكنك أن تبعث قيم للتعامل معها من خلال الدالة .

وإليك الآن مثالاً عن كيفية كتابة دالة لجمع عددين :

```
int sum ( int x , int y )
{
	Console.WriteLine("Value Of Var X Is : {0}",x);
	Console.WriteLine("Value Of Var Y Is : {0}",y);
	return x+y ;
}
```




في المثال السابق قمنا بإرسال قيمتين من خلال إستدعائها في الدالة الرئيسية Main Function والتعامل معها بواسطة طباعتهما وإرجاع قيمة حاصل جمعهما .

أما بالنسبة للإستدعاء فهنا يكمن المغزى وأريدك أن تركز معي قليلاً هنا .
لكي نستدعي دالة معينة يجب علينا ولكي نطبق مبدأ البرمجة الكائنية أن نستدعيه من خلال كائن من نفس الكلاس الموجود فيه الدالة الرئيسية لذلك نقوم بإنشاء الجملتين التاليتين :

```
Class1 c1 = new Class1();
a = c1.sum(1,3);
```

على فرض أن إسم الكلاس الذي نتعامل معه Class1 . هنا قمنا بإنشاء متغير من نفس نوع الكلاس الرئيسي للبرنامج ومن ثم قمنا بإستدعاء الدالة (جمع العددين) بإستخدام متغير الكلاس الجديد وبهذه الخطوة تقدمت السي شارب لتصبح لغة داعمة للبرمجة الكائنية Full OOP ، ولاحظ معي هنا أننا قمنا بإرسال قيمتين للدالة وهما 1 والعدد 3 وإسناد قيمة الجمع للمتغير الذي يحمل نفس نوع الدالة وهو المتغير a .
و إليك البرنامج كاملاً :

```
using System;

namespace First_Application_With_Console
{
      class Class1
      {
            [STAThread]
            static void Main(string[] args)
            {
                  int a ;
                  Class1 c1 = new Class1();
                  a = c1.sum(1,3);
                  Console.WriteLine("Value Of Sum X & Y Is : {0}",a);
            }

            int sum ( int x , int y )
            {
                  Console.WriteLine("Value Of Var X Is : {0}",x);
                  Console.WriteLine("Value Of Var Y Is : {0}",y);
                  return x+y ;
            }
      }
}
```

فكما تلاحظ راعينا بعض النقاط وهي :
1 – إنشاء دالة لجمع عددين وقمنا بإفتراض أن لديها وسيطين لعملية الجمع .
2 – إرجاع نفس نوع الدالة حيث أن جمع عددين صحيحين هو عدد صحيح وهو نفس نوع الدالة .
3 – قمنا بإنشاء كائن يحمل نوع الكلاس الذي يحمل الدالة الرئيسية .
4 – قمنا بإستدعاء الدالة بواسطة كائن من نوع الكلاس الأساسي وقمنا بإرسال الوسائط اللازمة والمطلوبة من قِبل الدالة .
5 – قمنا بإسناد قيمة الدالة المسترجعة بمتغير يحمل نفس نوع الدالة .

ولنتكلم الآن قليلاً من التفصيل عن هيكلية الدالة ، فهي غالباً تكون من نوع Void ومعناها أنها لا تعيد أي قيمة وهي هنا تريحنا من جملة Return الموجودة في آخر الدالة .

ولو نظرت إلى الدالة الرئيسية و هيكلها لوجدتها عبارة عن دالة من النوع Void لذلك لم نحتاج لجملة Return . فيمكنك تغيير نوعها إلى أي نوع ولكن يجب أن ترجع منها قيمة توافق النوع الذي حددت . كذلك تلاحظ هنا وجود وسائط هنا في الدالة الرئيسية وقد تحدثنا عنها في الدرس الثالث .

وأيضاً تلاحظ كلمة Static وهي هنا تعني أنك يا برنامج يجب أن تحفظني عندك في الذاكرة أنني أنا الدالة الرئيسية من خلال الكلمة (Main) ومن خلال كلمة ستاتيك الدالة على حجز المكان للدالة الرئيسية حتى نهاية البرنامج ولاحظ معي لو أنك قمت بحذف كلمة Static لحدث عندك خطأ رئيسي يقول لك أنه لا يوجد عندك منطقة معرفة للدالة الرئيسية لذلك لا أستطيع قراءة البرنامج ككل .

بمعنى أنك عندما تعرف متغير في دالة معينة من النوع العادي (إستخدام الأنواع الموجودة فقط) عند الوصول إلى القوس النهائي للدالة (}) فإنه سيقوم بحذفه من الذاكرة وكأنه غير موجود ولكن إذا كان من النوع الإستاتيكي فإنه سيحافظ على قيمته حتى الإنتهاء من البرنامج .
وقد شرحت قبل ذلك معلومات كافية عن النوع الإستاتيكي .

وفي الدالة السابقة في برنامج الجمع لم تكن الدالة معروفة لدى البرنامج لذلك قمنا بإستدعائها بواسطة كائن من نفس نوع الكلاس الأساسي .

فلو أنك قمت بإضافة كلمة Static للدالة الخاصة بالجمع لكان بإمكانك إستخدام الدالة صريحة بدون وسيط وكائن من نوع الكلاس ولتفهم ما قلت إليك المثال السابق مع بعض التعديلات عليه :

```
using System;

namespace First_Application_With_Console
{
	class Class1
	{
		[STAThread]
		static int Main(string[] args)
		{
			int a ;
			a = sum(1,3);
			Console.WriteLine("Value Of Sum X & Y Is : {0}",a);
			return 0 ;
		}

		static int sum ( int x , int y )
		{
			Console.WriteLine("Value Of Var X Is : {0}",x);
			Console.WriteLine("Value Of Var Y Is : {0}",y);
			return x+y ;
		}
	}
}
```

ماذا تلاحظ في المثال السابق ؟
قمنا بجعل الدالة إلى النوع الإستاتيكي و إستدعيناها صريحة بدون وسائط من خلال أي متغير .

فهنا آمل أن تكون قد فهمت معنى الإستاتيك وكيفية التعامل معه من خلال الدوال .

حسناً لنأتي الآن للمتغير الإستاتيكي :
فكما قلت لك بإمكانك تعريف دالة أو حتى متغير من النوع الإستاتيكي بواسطة الكلمة المحجوزة Static كما في المثال التالي :

```
static Type Var = Value ;
static Type Var ;
```

أيضاً يمكنك كتابة المتغيرات في قسم الكلاس أي بين الكلاس الرئيسي وبين الدوال المختلفة ولو سألتك عن كيفية الوصول إليه فهل تستطيع الإجابة ؟
ستقول لي بما أنه موجود في قسم الكلاس الرئيسي يمكن التعامل معه في الدالة بشكل عادي أي بكتابة إسم المتغير مباشرةً كما في المثال التالي :

```
class Class1
{
	int x = 10;
	[STAThread]
	static void Main(string[] args)
	{
		Console.WriteLine(x);
	}
}
```

وسأقول لك يا صاحبي أن كلامك غير صحيح .



فهنا أيضاً يجب عليك مراعاة نوع المتغير ، فإذا كان متغير من الأنواع العادية وليس إستاتيكياً وقمت بإستعماله مباشرة في الدالة فسينتج خطأ والحل لذلك كما في الدالة وهي إنشاء متغير جديد يحمل إسم الكلاس الأساسي ثم التعامل معه كالمثال التالي :

```
class Class1
{
	int x = 10;
	[STAThread]
	static void Main(string[] args)
	{
		Class1 c1 = new Class1();
		Console.WriteLine(c1.x);
	}
}
```

لاحظ أننا قمنا في المثال السابق بتعريف متغير جديد لأن المتغير x عبارة عن متغير غير إستاتيكي .
ولكن .. قد تسأل هل الأمر بالنسبة للمتغير كما هي في الدالة ؟
نعم يا صديقي العزيز كما في الدالة فإذا قمت بتعريف المتغير كنوع إستاتيكي تتخلص من مشكلة تعريف متغير من نوع الكلاس الأساسي لأن البرنامج سيتعرف عليه طيلة تنفيذ البرنامج ليصبح معك البرنامج كالتالي :

```
class Class1
{
	static int x = 10;
	[STAThread]
	static void Main(string[] args)
	{
		Console.WriteLine(x);
	}
}
```

ويمكن أن ألخص لك النقاط الأربعة للدالة كالتالي :

1 – إذا كان عندك دالة من النوع العادي وتريد إستدعاء دالة ذات نوع إستاتيكي يمكنك إستدعائها بشكلها الصريح لأنها معروفة لدى البرنامج كالمثال التالي :

```
void f1()
{
	f2();
}

static void f2()
{
	Console.WriteLine("Hello");
}
```

2 – إذا كان عندك دالة من النوع العادي وأردت أن تستدعيها من دالة أخرى تحمل النوع العادي أيضاً يمكنك إستدعائها مباشرةً بذكر إسمها بشكل صريح كما في المثال التالي:

```
void f1()
{
	f2();
}

void f2()
{
	Console.WriteLine("Hello");
}
```




3 – إذا كانت عندك دالة من النوع الإستاتيكي وأردت إستدعاء دالة تحمل النوع العادي يمكنك إستدعائها لأنها معروفة لدى البرنامج كالمثال التالي :

```
static void f1()
{
      Class1 c1 = new Class1();
      c1.f2();
}

void f2()
{
      Console.WriteLine("RTAQ");
}
```

4 – إذا كانت عندك دالة من النوع الإستاتيكي وأردت إستدعاء دالة أخرى من النوع الإستاتيكي أيضاً يمكنك كتابة الدالة صريحة وبدون وسائط لأنها معروفة لدى البرنامج كالتالي :

```
static void f1()
{
      f2();
}

static void f2()
{
      Console.WriteLine("OMS");
}
```

إذن الإستدعاء بالشكل الصريح كالتالي :

Static → Standard x
Static → Static ✓
Standard → Static x
Standard → Standard ✓

النوعان إذا كانا متشابهان يجوز مناداتهما بشكل صريح داخل بعض

أما بالنسبة لإستدعاء المتغير داخل الدالة :
1 – إذا كان المتغير من النوع العادي والدالة من النوع العادي يمكنك إستدعاءه بشكل صريح كالتالي :

```
int x = 4 ;
void Fun1()
{
      Console.WriteLine(x);
}
```

2 – إذا كان المتغير من النوع العادي والدالة من النوع الإستاتيكي يجب عليك تعريف متغير من نوع الكلاس الأب كما في المثال التالي :

```
int x = 4 ;
static void Fun1()
{
      Class1 aa = new Class1();
      Console.WriteLine(aa.x);
}
```

3 – إذا كان المتغير من النوع الإستاتيكي والدالة من النوع العادي يمكنك إستدعاءه بشكل صريح كالتالي :

```
static int x = 4 ;
void Fun1()
{
      Console.WriteLine(x);
}
```

4 – إذا كان المتغير من النوع الإستاتيكي والدالة من النوع الإستاتيكي يمكنك إستدعاءه بشكل صريح كما في المثال التالي :

```
static int x = 4 ;
static void Fun1()
{
      Console.WriteLine(x);
}
```

إذن للإستدعاء بشكل صريح كالتالي بحيث يكون المتغير (الطرف الأيسر) والدالة (الطرف الأيمن) :

Static → Standard ✔
Static → Static ✔
Standard → Static x
Standard → Standard ✔

أرجوا أن تكون قد وعيت إستخدامات النوع الإستاتيكي مع الدوال والمتغيرات .

الكلمة المحجوزة This :
كما شاهدت في الأنواع للمتغيرات والدوال قد تضطر أحياناً لتعريف كائن من نفس نوع الكلاس . هنا تقوم لغة السي شارب بتقديم متغير جاهز وقد وضع في مرتبة الكلمات المحجوزة والذي يدل على إسم الكلاس الأب . فمثلاً بدلاً من كتابة جملتين كالتالي :

```
Class1 aa = new Class1();
Console.WriteLine(aa.x);
```

سهلت عليك وقالت لك خذ وأكتب الجملة بسطر واحد فقط و بإستخدام الكلمة المحجوزة this :

```
Console.WriteLine(this.x);
```

ولكن هنا يجب التنبيه في أنه ليس بإمكانك إستخدام هذه الكلمة المحجوزة في دالة من النوع الإستاتيكي مثلاً مثل الدالة الرئيسية فيحظر عليك فعل ذلك . كذلك يمنع إستعمالها مع متغير من النوع الإستاتيكي أيضاً والسبب في ذلك أن البرنامج نفسه يعرف المتغير أو الدالة ذات النوع الإستاتيكي فلا داعي لإستخدام هذه الكلمة .

الآن سأقوم بإعطائك بعض الأسئلة :
(جاوب عليها وأرسل الجواب على الإيميل الخاص بي مع ذكر الإسم وضع عنوانها HW_C#_5)

1 – أكتب برنامج للطلب من المستخدم بإدخال قيمتين ثم قم بحساب حاصل جمعهما وحاصل ضربهما ومقسوم الأول على الثاني (راعي أصفار المقام) بإستخدام دوال خارج الدالة الرئيسية وقم بحفظ ناتج كل عملية في مصفوفة مكونة من 3 عناصر وقم بطباعتهم من خلال الدالة الرئيسية .

2 – حدد الناتج (دون إستعمال الحاسوب) في كل برنامج من البرامج التالية مع الشرح خلال التتبع :

I -
```
static void Main(string[] args)
{
      int i = 3 ;
      OMS:
      if ( i++ > 6 )
            goto Finish;

      for (  ; i++ < 9 ; )
      {
            if ( i < 3 )
                  goto OMS;
            Console.WriteLine(i+1);
      }
      Finish:
      Console.WriteLine("Finish With I Is : {0}",i);
}
```




II –

```
class Class1
{
	enum RTAQ{Adil = 1 , OMS = 0 , Tarek = 3 , Islam = 2};
	[STAThread]
	static void Main(string[] args)
	{
		RTAQ r1 ;
		for ( int i = 0 ; i < 4 ; i++)
		{
			r1 = (RTAQ)i;
			Console.WriteLine(r1);
		}
	}
}
```

3 – قم بكتابة برنامج لزيادة متغير من نوع عادي ومن ثم طباعته في دالة إسمها Change من نوع void وهي من النوع الإستاتيكي حيث المتغير موجودة في قسم ما في داخل الكلاس الرئيسي .

4 – حدد الخطأ في كل جملة من الجمل التالية هل هو إملائي أم منطقي وبدون إستخدام الحاسوب :

I –

```
 int i = 3 ;
for (  ; j++ < 9 ; )
{
	Console.WriteLine(i+1);
}
```

II –

```
 static void Main(string[] args)
{
	for ( int i = 0 ; i < 56 ; i++ )
	{
		Console.WriteLine(i+1);
		goto Lab :
	}
	lab ;
	    Console.WriteLine("Finish");
}
```

III –

```
	enum RTAQ{Adil = 1 , OMS = 6 , Tarek = 3 , Islam = 2};
	[STAThread]
	static void Main(string[] args)
	{
		int x = 4 ;
		RTAQ r1 = new RTAQ();
		r1 = (RTAQ)x;
		Console.WriteLine(r1);
	}
```

IIII –

```
static void Main(string[] args)
{
	int x = 4 ;
	char y = 6 ;
	Console.WriteLine("The Sum Is : {0}",x+y);
}
```

بسم الله الرحمن الرحيم

المحاضرة الثامنة :

مواضيع متقدمة حول الدوال – قسم الـ Console :

تحدثنا في البدايات أنه بإمكانك إستخدام الباراميتر الموجودة ضمن الدالة الرئيسية ولنتحدث الآن عن كيفية التعامل مع هذه الباراميتر :
فلو قمنا بتنفيذ الملف التنفيذي exe المرفق في البرنامج وقمنا بإضافة باراميتر كيف يمكننا التعامل معه ؟ وكيف سنعلم أنه أدخل أو أنه لم يدخل وسائط ؟! .
إليك الآن مثالاً ويليه الشرح بالتفصيل :

```
using System;
class Class1
{
	static void Main(string[] args)
	{
		int Sum = 0 ;
		if ( args.Length != 0 )
			for ( int i=0 ; i<args.Length ; i++ )
				Sum+=int.Parse(args[i]);
		Console.WriteLine(Sum);
	}
}
```

البرنامج السابق يقوم بحساب مجموع القيم المدخلة . ولكن إذا قمت بتنفيذ البرنامج فسيطبع لك القيمة 0 كيف ذلك ؟! حسناً لا تنسى أننا في البداية قلنا (الوسائط) وهنا يقوم بتنفيذ البرنامج من غير وسائط . والآن كيف يمكننا إدخال وسائط ؟
بعد تنفيذ البرنامج قم بالذهاب إلى المجلد الأصلي للمشروع ثم ستجد هناك مجلد إسمه Bin ثم تجد فيه مجلد وحيد وهو Debug قم بنسخه إلى موقع سهل مثل (C:\ ) سنقوم الآن بإسناد الوسائط له والآن إذهب إلى Run → Start وقم بكتابة (cmd) إذا كنت تمتلك ويندوز إكس بي أو (Command) إذا كنت تمتلك غيره .
إذهب إلى الـ Directory الأصلية بواسطة الجملة (cd\ ) ثم قم بإستدعاء برنامجك وليكن إسمه Sum.exe كالتالي :

```
C:\>sum 1 3 4
8
C:\>
```

كما لاحظت في المثال السابق كل القيم تم تخزينها في مصفوفة من النوع String لأن المدخلات دائماً عبارة عن نص .

الكلاس :
كما تلاحظ في كل مشروع أنه يوجد كلاس رئيسيي . والسؤال هنا هل يمكننا كتابة أكثر من كلاس وهل يجوز كتابته داخل كلاس آخر ؟

بالنسبة للكلاس الأب فغالباً يكون على الهيئة البسيطة كما في الكود التالي :

```
using System;
class Class1
{
	static void Main(string[] args)
	{

	}
}
```

حيث يوجد به دالة رئيسية ، والهدف من بتعريف كلاس داخل كلاس كالتالي :
فلو كان عندك دوال تتعلق بموضوع ما مثلاً بالرياضة ودوال أخرى تتعلق مثلاً بالبرامج الحاسوبية فمن الأصح وضع كل موضوع في كلاس خاصة به وفائدة ذلك . هو عندما نقوم بإنشاء مكتبة من دوال قمنا بإنشائها في مشروع آخر ، سنعلم هنا أن كل الدوال المتعلقة بالكلاس موجودة داخله . ولاحظ أن كل كلاس يمكن أن يحتوي على الدالة الرئيسية التي سيبدأ البرنامج منها .

إذن يجوز لنا إستخدام كلاس آخر في نفس البرنامج كالتالي :

```
using System;
class Class1
{
	static void Main(string[] args)
	{

	}
}
class Class2
{
	static void Main(string[] args)
	{

	}
}
```

ولكن ... لا تستعجل في تنفيذ البرنامج لأنه سيظهر عندك خطأين وهما أنك لم تحدد من الذي سيبدأ أولاً فهنا عندما تقوم بتنفيذ البرنامج يجب عليك أن تستخدم إحدى الكلاسات الموجودة عندك كأساس وأن تهمل الثاني لأنه لن يبدأ به ، وحتى نقوم بتفضيل كلاس على آخر إليك خطوات ذلك :

إختر مشروعك من خلال الرقم 1 ثم قم بإختيار الخصائص من خلال الرقم 2 أو من خلال الزر الأيمن للماوس ومن ثم إختر خصائص ، ثم قم بإختيار عام كما في الرقم 3 ثم إذهب إلى إقلاع الكائن كما في الرقم 4 ثم قم بإختيار أي كلاس من الكلاسات الموجودة عندك كما في الرقم 5 ثم قم بالضغط على موافق وهنا يقوم بالبدء من الكلاس الذي حددته .



كذلك يمكنك كتابة كلاس داخل كلاس كما في المثال التالي :

```
using System;
class Class1
{
	static void Main(string[] args)
	{

	}
	class Class2
	{
		static void Main(string[] args)
		{
			Console.WriteLine("Welcome");
		}
	}
}
```

ولكن هنا يجب عليك تحديد أي كلاس ستبدأ به فبإمكانك أن تبدأ بالكلاس الداخلي ولن يكون للخارجي علاقة به . وأيضاً يمكنك تحديده كما في الخطوات السابقة .

بالنسبة للدوال فهناك أربعة أنواع حسب النوع وطريقة الإرجاع كالتالي :

1 – النوع الأول لا يأخذ قيمة (وسائط) ولا يرجع قيمة :

ويمكن تسميته بالعقيم أي أنه لا يحتاج باراميتر ولا يرجع أي قيمة وهو من النوع Void .

وإليك مثالاً على هذا النوع :

```
using System;
class Class1
{
	static void Main(string[] args)
	{
		Fun_Type_1();
	}

	static void Fun_Type_1()
	{
		Console.WriteLine("Welcome In First Type Of Function .");
	}
}
```

وهنا وظيفة هذا النوع غالباً للطباعة فهو ينفذ أمور ليس لها علاقة بالدالة الرئيسية ، وطريقة إستدعائه بكتابة إسمه مباشرة .

ملاحظة : كتبتُ قبل الدالة كلمة Static للتخلص من تعريف كائن من نوع الكلاس وإذا لم تفهم ما قلت راجع الدرس الذي قبله .

2 – النوع الثاني يأخذ قيمة أو أكثر ولا يرجع قيمة :

هنا يقوم بأخذ باراميتر من الدالة الرئيسية لحساب أو إجراء عمليات داخل هذه الدالة ولا يقوم بإرجاع أي قيمة للدالة الرئيسية ، وإليك مثالاً على ذلك :

```
using System;
class Class1
{
	static void Main(string[] args)
	{
		int x = 6 ;
		int y = 8 ;
		Fun_Type_2(x,y);
	}

	static void Fun_Type_2(int x,int y)
	{
		Console.WriteLine("The Sum Is : {0}",x+y);
	}
}
```




لاحظ أننا في هذا النوع قمنا بإستدعاءه من الدالة الرئيسية وكتبنا الباراميتر التي يأخذها .

3 – النوع الثالث لا يأخذ قيم و يرجع قيمة :
وهو غالباً يكون من الأنواع الأخرى غير النوع Void ولاحظ أننا لا نقوم بأخذ وسائط من الدالة الرئيسية أن أننا غالباً نضع معايير لهذه الدالة وحسب الطلب نقوم بإستخدامها ولكي تفهم ما أقول إليك المثال التالي :

```
using System;
class Class1
{
	static void Main(string[] args)
	{
		int radius = 3 ;
		Console.WriteLine("The Area Of Cycle Of Radius {0} Is :
		                   {1}",radius, 4*Fun_Type_3()*radius);
	}

	static float Fun_Type_3()
	{
		// PI
		return 3.14f ;
	}
}
```

هنا في المثال السابق قمنا بحساب مساحة كرة بواسطة القيمة Pi الموجودة في دالة ، حيث أن هذه القيمة هي قيمة ثابتة فلا داعي لإرسال الوسائط والتغيير فيها ، كذلك لاحظ أننا قمنا بطباعة الدالة فوراً لأنها تقوم بإرجاع قيمة ، كذلك يمكنك إسنادها ووضع قيمتها في متغير ثم تقوم بطباعة هذا المتغير كالتالي :

```
int radius = 3 ;
float PI = Fun_Type_3() ;
Console.WriteLine("The Area Of Cycle Of Radius {0} Is : {1}",radius,
                   4*PI*radius);
```

4 – النوع الرابع يأخذ قيم و يرجع قيمة :
وفي هذا النوع نقوم بإرسال وسائط ونقوم بحسابات معينة لها بواسطة معاملات حسب الطلب ولكي تفهم ذلك إليك المثال التالي :

```
using System;
class Class1
{
	static void Main(string[] args)
	{
		int x = 15 ;
		int y = 8 ;
		int Sum = Fun_Type_4(x,y) ;
		Console.WriteLine("The Sum Of {0} & {1} Is : {2}",x,y,Sum);
	}

	static int Fun_Type_4(int x , int y)
	{
		return x+y ;
	}
}
```

قمنا بإرسال قيمتين وأرجعنا مجموعهما .

ملاحظة :
في النوعين الأخيرين يجب أن تسند القيمة الراجعة إلى متغير أو تقوم بالطباعة بشكل فوري . عكس النوعين الأولين فهناك يمكنك طباعة إسم الدالة لوحدها وبدون إسناد في سطر لوحدهما .

ولكن ... ماذا بالنسبة للخروج الإضطراري من الدالة ؟

كما تعلمنا سابقاً أنه بإمكانك الخروج من حلقات التكرار بواسطة الجملة Break ولكن هنا الأمر مختلف ، فلكي تقوم بالخروج الإضطراري من الدالة تستخدم كلمة Return حتى ولو كان نوع الدالة Void أي أنها لا ترجع أي قيمة ، و هاك مثال على ذلك :

```
using System;
class Class1
{
	static void Main(string[] args)
	{
		char x =char.Parse(Console.ReadLine());
		Console.WriteLine(Dont_5(x));
	}

	static char Dont_5(char x)
	{
		if ( x == '5' )
		{
			Console.WriteLine("Oh Dear You Cannot Use Number 5 ...");
			return ' ' ; // Break
		}
		return x ;
	}
}
```

لاحظ في المثال السابق سيقوم المستخدم بإدخال قيمة غير القيمة 5 وإذا قام بإدخالها سيقوم البرنامج بالخروج من الدالة مع ظهور رسالة تقول لك أنك قمت بإدخال الرقم 5 المحظور عليك إدخاله .

بسم الله الرحمن الرحيم

المحاضرة التاسعة :

الوسائط في الدوال و إكتشاف الأخطاء – قسم الـ Console :

لعلك لاحظت على أنني أركز على الدوال كثيراً ، والسبب في ذلك أن كفاءة البرنامج تحدد بواسطة تقسيم البرنامج إلى دوال فرعية كل دالة تقوم بعمل محدد وذلك لسهولة إكتشاف الأخطاء ، وإذا رغبت في تغيير أمر لا تضطر لتغيير جزري في البرنامج ككل .

حسناً في الدروس السابقة قمنا بإرسال متغيرات إلى الدوال بوسائل شتى ، والسؤال الآن ماذا لو أردت أن أرسل مصفوفة كوسيط ؟ وكيف أفعل ذلك ؟

إليك المثال التالي ثم نجاوب على الأسئلة السابقة :

```
using System;
class Class1
{
	static void Main()
	{
		int []Q =new int[4]{1,2,3,4};
		Fun(Q);
	}

	static void Fun(int[] Q)
	{
		int Sum=0 ;
		for (int i=0 ; i<Q.Length;  i++ )
			Sum+= Q[i] ;
		Console.WriteLine("The Sum Is : {0}",Sum);
	}
}
```

في المثال السابق قمنا بالتالي :

1 – تعريف مصفوفة من النوع الصحيح وإسناد قيم أولية لعناصرها .
2 – كتابة دالة لجمع محتوى عناصر المصفوفة .
3 – إدخال جملة For لأخذ محتوى عناصر المصفوفة ويمكنك أيضاً إستخدام الجملة foreach .
4 – لاحظ أننا قمنا بتعريف مصفوفة داخل الدالة في قائمة الوسائط .
5 – قمنا بإرسال إسم المصفوفة فقط إلى الدالة للتعامل معها .

حسناً إليك الشرح الآن :

قلنا سابقاً أنه يجوز إرسال متغيرات إلى الدوال والتعامل معها ، ولم يختلف الأمر كثيراً هنا فقد قمنا بإرسال إسم المصفوفة وكأنها متغير ثم قمنا بتعريف المصفوفة ونوعها في الدالة ، ويجب مراعاة النوع هنا فلكي ترسل مصفوفة من النوع الصحيح يجب عليك كتابة النوع أيضاً في الدالة وهو هنا العدد الصحيح ، قمنا في الدالة بالتعامل معها وقد تعاملنا مع مصفوفة كأنها موجودة أصلاً في الدالة الرئيسية .

ملاحظة :

هل يجوز كتابة إسم البراميتر غير الإسم المرسل ؟ يعني هل أستطيع أن أكتب إسم المصفوفة في الدالة غير حرف الـ Q ؟

الجواب هنا نعم يمكنك التغيير ولكن ! من الإحترافية Professional أن تكتبها كما هي في الدالة التي إستدعيتها منها .

حسناً الآن ماذا لو أنني أردت أن أبعث قيم فقط من غير مصفوفة وأريد من الدالة التعامل مع القيم المرسلة كمصفوفة ومن غير حجم معين ، يعني أن أبعث قيم كما أشاء إلى دالة معينة !

هنا يجب عليك تعريف مصفوفة في الدالة ولا يجب عليك إرسال مصفوفة من مكان الإستدعاء . ولكي تفهم وتناقش ما قلت إليك المثال التالي :

```
using System;
class Class1
{
	static void Main()
	{
		Fun(3,5,12,3);
	}

	static void Fun(params int[] Q)
	{
		int Sum=0 ;
		for (int i=0 ; i<Q.Length;  i++ )
		      Sum+= Q[i] ;
		Console.WriteLine("The Sum Is : {0}",Sum);
	}
}
```

لاحظ معي هنا أننا قمنا بإرسال قيم مباشرة من الدالة الرئيسية ولم نحدد عددها وقمنا بالتعامل معها كمصفوفة في الدالة .

ولكن هنا كلمة جديدة وهي params فيا هل ترى ما معناها ؟
هنا كتبناها لإعطاء حجم غير محدد للمصفوفة الموجودة بعد الكلمة تماماً ، يعني أن الحجم هو عدد مفتوح ويتم تحديده من مكان إستدعاء الدالة .

إرسال القيم By reference :
هل فكرت في إرسال قيمة إلى دالة معينة وأردت أن تجري عليها الدالة تغييرات وتنعكس معك بعد إستدعائها ؟ يعني أن تغير قيمة بواسطة دالة !
هنا تتيح لنا لغة السي شارب وكغيرها من اللغات إرسال المتغيرات وإجراء التعديلات عليها ، فقد كنا سابقاً نبعث المتغيرات ولا تؤثر عليها أبداً وكانت تسمى هذه الطريقة By Value أي أننا نبعث القيمة فقط ولا يتم إجراء أي تعديلات عليها .

في الإصدارات القديمة مثل Visual C++ 6.0 كان بالإمكان التعامل مع ذلك بواسطة المؤشرات (بوينتر) Pointer ولكن في السي شارب ألغيت هذه الفكرة تماماً ولم يعد هناك Pointer .
ولكي تبعث متغير وتجري علية التغييرات إليك المثال التالي :

```
using System;
class Class1
{
	static void Main()
	{
		int x = 5;
		Fun(ref x);
		Console.WriteLine("The New Value Of X Is : {0}",x);
	}

	static void Fun(ref int x)
	{
		x = 2*x;
	}
}
```

لاحظ أننا قمنا باستخدام الكلمة المحجوزة `ref` **إختصار كلمة Reference ومعناها التغيير على الباراميتر .**
ويجب علينا أيضاً أن نكتبها في مكان الإستدعاء وفي الدالة نفسها .
وإذا نفذت البرنامج السابق سينتج الرقم 10 بواسطة طباعة المتغير في الدالة الرئيسية .

والآن إلى قضية جديدة وهي ، ماذا لو أردت أن أعطي المتغير القيمة الأولية في دالة معينة ؟ بمعنى ماذا لو أردت إعطاء أول قيمة للمتغير من خلال دالة ؟

نحن نعرف أنه لا يجوز التعامل مع متغير وإجراء العمليات عليه إلا بعد إعطائه قيمة ، فكيف سننفذ الفكرة السابقة ؟
تتيح لنا لغة السي شارب الكلمة المحجوزة out والتي تعني أنني سأعطي قيمة لمتغير خارج الدالة الرئيسية وإليك تبياناً لهذه الكلمة في المثال التالي :

```
using System;
class Class1
{
	static void Main()
	{
		int x ;
		Fun(out x);
		Console.WriteLine(x);
	}

	static void Fun(out int x)
	{
		x = 5;
	}
}
```

فلو أنك قمت بحذف كلمة Out من الدالة ومن مكان إستدعائها لظهر عندك الخطأ المعروف وهو أنه لا يمكنك أن تبعث قيمة .
وكما الحال في الكلمة المحجوزة Ref فإنه يجب عليك أن تضع جملة Out في مكان الإستدعاء وفي مكان إستقبال البراميتر .

الخطأ Exception و إكتشافه بواسطة Try والجملة الرديفة Catch :
ماذا لو أننا كنا نشك في جملة أن ناتجها عبارة عن خطأ ، هل نترك مجال للخطأ أن يخرجنا بالقوة من البرنامج ؟
إذن لابد من إكتشاف الأخطاء ومنع حدوثها وإذا كان لابد فيجب معرفة المعالجة لهذا الخطأ .

هنا تتيح لنا لغة السي شارب إكتشاف الأخطاء بالجملة Try وهنا يجب وضع الجمل التي نشك أنه سيحدث خطأ منها في داخل هذه الجملة كالتالي :

```
try
{
 Statement ;
}
```

ويجب أن تتبعها جملة Catch أو جملة Finally .
ولهذه الجمل خمس حالات كالتالي : (ولنأخذ خطأ القسمة على صفر)

الحالة الأولى :
وهو أن تأتي جملة Try ثم نتبعها بجملة Catch كالتالي :

```
using System;
class Class1
{
	static void Main()
	{
		int x = 5 ;
		int y = 0 ;
		int z ;

		try
		{
			z = x/y ;
		}
		catch
		{
			Console.WriteLine("Error : Divide By Zero !!");
		}
	}
}
```

في الحالة السابقة عرفنا أنه سيحدث خطأ القسمة على صفر ووضعنا الجملة التي تؤدي إلى خطأ بين جملة Try وأتبعناها بجملة واحدة من Catch .

الحالة الثانية :
وهي وضع مجموعة الجمل التي تؤدي أخطاء ولكننا لا نعرف ما هو الخطأ ونتبعها بجملة Catch ولكن نعرف كائن من نعو الخطأ ونطبع ما هو الخطأ كالتالي :

```
using System;
class Class1
{
	static void Main()
	{
		int x = 5 ;
		int y = 0 ;
		int z ;

		try
		{
			z = x/y ;
		}
		catch ( Exception e )
		{
			Console.WriteLine(e.Message);
		}
	}
}
```

لاحظ سابقاً أننا أخذنا كائن من نوع الخطأ وقمنا بإستعماله .

الحالة الثالثة :
إستعمال الجملة Try ولكن بأكثر من جملة Catch وذلك بكتابة الخطأ كالتالي :

```
using System;
class Class1
{
	static void Main()
	{
		int x = 5 ;
		int y = 0 ;
		int z ;

		try
		{
			z = x/y ;
		}
		catch ( DivideByZeroException )
		{
			Console.WriteLine("Err : Divide By Zero");
		}
		catch ( OverflowException )
		{
			Console.WriteLine("Err : Over Flow");
		}
	}
}
```

ويجوز لك إختيار العدد المناسب من جمل Catch كما تحتاج .

الحالة الرابعة :
وهي إستخدام جملة Finally وهي جملة سيمر عليها البرنامج سواء أكان هناك خطأ أم لم يكن .

وإليك مثال ذلك :

```
using System;
class Class1
{
	static void Main()
	{
		int x = 5 ;
		int y = 1 ;
		int z ;

		try
		{
			z = x/y ;
		}
		finally
		{
			Console.WriteLine("Hello");
		}
	}
}
```

لاحظ في المثال السابق أنها تعتبر كمثابة جملة بعد حدوث الخطأ .

الحالة الخامسة :

وهي إستخدام جملة Catch مع جملة Finally وهي جامعة للأنواع الأربعة السابقة وإليك مثال على هذه الحالة :

```
using System;
class Class1
{
	static void Main()
	{
		int x = 5 ;
		int y = 0 ;
		int z ;

		try
		{
			z = x/y ;
		}
		catch(DivideByZeroException)
		{
			Console.WriteLine("Err : Divid By Zero");
		}
		catch ( Exception e)
		{
			Console.WriteLine(e.Message);
		}
		finally
		{
			Console.WriteLine("There An Error !!");
		}
	}
}
```

لاحظ هنا أنه سيمر على جملة واحدة من جمل الـ Catch وسيمر إجبارياً على جملة Finally .

ولكن .... ماذا لو أنك أردت حدوث خطأ في برنامجك ؟؟
بالأصح ماذا لو كان برنامجك لا يتعامل مع الرقم 2542 مثلاً بالشكل المطلوب ورأيت أن تلصق فيه خطأ للتخلص من مشكلته ؟؟

هنا يجب عليك أن ترمي عليه خطأ وإلا سيحدث عندك خطأ معنوي أي أن البرنامج لن يظهر لك أي خطأ ولكن الخطأ سيؤثر تأثيراً كبيراً من الناحية المنطقية .



راعت لغة السي شارب هذه النقطة وقدمت لنا كلمة Throw كالتالي :
عند حدوث أو الوصول للرقم المطلوب قم برمي خطأ عليه كالتالي :

```
using System;
class Class1
{
	static void Main()
	{
		for ( int i=0 ; i < 20 ; i++ )
		{
			if ( i == 13 )
				throw ( new Exception()) ;
			Console.WriteLine ( i ) ;
		}
	}
}
```

قم بتنفيذ البرنامج السابق ولاحظ أنه سيعطيك خطأ عند الرقم 13 يمكنك إزالته بواسطة الجملة Try فهذا أمر راجع لك لأنك أنت إخترت الخطأ .



بسم الله الرحمن الرحيم

المحاضرة العاشرة :

مكتبة التعامل مع النصوص String – قسم الـ Console :

أتاحت لنا لغة السي شارب نوع جديد وهو النوع النصي String وهو هنا يعتبر نوح بحد ذاته إضافة إلى وجود كلاس كامل للعمليات التي نجريها عليها مثل الإضافة والمقارنة وغيرها وهو كلاس String بداية حرف كبير عكس النوع وهو بداية حرف صغير ويصبح لونه أزرق (يعني كلمة محجوزة) . كانت في الإصدارات وغيرها من اللغات السابقة لا تتعامل بكفاءة مع هذا النوع فكنا آن ذاك نستخدم مصفوفة من متغير خاني ونقوم بملئها .

سنتعرف الصيغة العامة لتعريف المتغير والعمليات المختلفة عليه :

```
string var = "Value";
```

لاحظ هنا وجود اللون الأزرق على النوع ، كذلك يمكنك إعطاء المتغير القيمة في سطر منفرد كالتالي :

```
string var ;
var = "Value" ;
```

هنا كثير من المبرمجين كان بإمكانهم أخذ خانة خانة بواسطة هذا الكود :

```
string Test ;
Test = "ABCDEFG" ;
Console.WriteLine(Test[5]);
```

وكان بإمكانهم أيضاً تغيير القيم لأي خانة كالتالي :

```
string Test ;
Test = "ABCDEFG" ;
Test[3] = "!";
Test[3] = '!';
```

لكن هنا الأمر مختلف فمن الممنوع إستخدام ذلك بل ومن الخطأ كذلك .
فهنا يمكنك أخذ خانة ووضعها في متغير خاني Char كما في الكود التالي :

```
string Test ;
Test = "ABCDEFG" ;
char f = Test[5];
Console.WriteLine(f);
```

وكذلك يمكنك الحصول على عدد الخانات التي تحتويها (الطول) بإستخدام الخاصية Length بعد تعريف لمتغير النصي كالتالي :

```
string Test ;
Test = "ABCDEFG" ;
Console.WriteLine("Number Of Char Is : {0}",Test.Length );
```

كذلك يمكنك نسخ نص إلى متغير نصي من خلال متغير آخر بإستعمال الدالة Copy كالتالي :

```
string Test,ABC ;
Test = "ABCDEFG" ;
ABC = String.Copy(Test);
Console.WriteLine(ABC );
```

لاحظ هنا أننا إستخدمنا كلاس الـ String وأخذنا منه دالة (Method) ولاحظ الفرق بين الكلاس والنوع .

ويمكنك إضافة أو وصل متغير نصي بمجموعة من الأحرف بإستعمال الدالة Concat كالتالي :

```
string Test,ABC ;
Test = "ABCDEFG" ;
Test += String.Concat('H',"I","JKLM");
Console.WriteLine(Test );
```

لاحظ هنا أننا قمنا بإستخدام كلاس الـ String .
ويمكنك فعل السابق بواسطة معامل (+) وهو هنا يعتبر Override لدالة Concat كالتالي :

```
string Test;
Test = "ABCDEFG" ;
Test += 'H' + "I" + "JKLM" ;
Console.WriteLine(Test );
```

وكذلك يمكنك إزالة الفراغ (Space) الموجود قبل وبعد الكلمة بواسطة الدالة Trim( ) كالتالي :

```
string Test;
Test = "  ABCDEFG  " ;
Test = Test.Trim();
Test += 'H' + "I" + "JKLM" ;
Console.WriteLine(Test );
```

ولجعل الحروف كبيرة أو صغيرة نستخدم الدالتان ToUpper و ToLower كالتالي :

```
string Test;
Test = "AbCdEfG" ;
Console.WriteLine(Test.ToUpper() + "\n" + Test.ToLower());
```

لاحظ أننا إستخدمنا الجملة "\n" وهو تعني New Line أي أنه سيقوم بإضافة سطر جديد .
فيمكنك هنا إضافتها للمتغير النصي كالتالي :

```
string Test ;
Test = "ABC\nDEF\nGHI" ;
Console.WriteLine(Test);
```

فيقوم بطباعة 3 أسطر هنا .
وهناك أيضاً مجموعة مثل هذه الشاكلة كالتالي :

Escape Sequence	**Represents**
\a	Bell (alert)
\b	Backspace
\f	Formfeed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\'	Single quotation mark
\"	Double quotation mark
\\	Backslash
\?	Literal question mark
ooo	ASCII character in octal notation
x*hhh*	ASCII character in hexadecimal notation

وهذا الجدول مأخوذ من ملفات المساعدة MSDN .



ويمكنك أيضاً إجراء عمليات المقارنة على متغيرين من نوع نصي كالتالي :

```
using System;

class Class1
{
	static void Main()
	{
		string TestA ;
		TestA = "ABC" ;
		Console.Write("Enter Your Text To Compare ( {0} ) : ",TestA);
		string TestB = Console.ReadLine() ;
		int Com = TestA.CompareTo(TestB) ;

		if ( Com == 0 )
			Console.WriteLine("{0} = {1}",TestA,TestB);
		else if ( Com == 1 )
			Console.WriteLine("{0} > {1}",TestA,TestB);
		else if ( Com == -1 )
			Console.WriteLine("{0} < {1}",TestA,TestB);
	}
}
```

ويمكنك إستنتاج أنه يجب أن يرجع قيمة من نوع عدد صحيح وهي إما أن تكون 1 أو 0 أو 1- ولذلك إذا كان الأول أكبر أو متساويين أو الثاني أكبر على التوالي .

وهنا يقوم المترجم بالمقارنة على أساس الأسكي كود ASCII CODE وهي أرقام مشفرة لكل خانة من خانات لوحة المفاتيح . ومع مراعاة أن لغة السي شارب هي لغة حساسة أي أن حرف الـ a غير عن حرف الـ A ولكل واحد منهما رقم آسكي خاص به .

وهنا يجب التنبيه إلى أمر وهو معامل المساواة (=) فإليك الجمل التالي :

```
using System;

class Class1
{
	static void Main()
	{
		string TestA,TestB ;
		TestA = "ABC" ;
		TestB = "!!!";
		Console.WriteLine(TestA == TestB);
		Console.WriteLine(TestA = TestB);
	}
}
```

ففي جملة الطباعة الأولى يقوم المترجم بطباعة متغير من نوع بولياني ومعناه هنا هل المتغير الذي على اليسار مساوي للذي على اليمين ؟ والجواب إما True ويعني صحيح أو False ويعني خطأ ، وأما الجملة الثانية فمعناها أنك يا مترجم قم بإسناد القيمة التي على اليمين إلى القيمة التي على اليسار وقم بطباعة المتغير الذي على اليسار .

Jordan , Amman Tel : (00 962 79 6527425) Omssd84@hotmail.com



بسم الله الرحمن الرحيم

المحاضرة الحادية عشر :

التعامل مع الملفات (الإدخال والإخراج) – قسم الـ Console :

من طرق التخزين المعروفة الملفات Files وقواعد البيانات Database وقد جاءت لغة السي شارب مكملة وداعمة لهذين العنصرين الهامين فقد أتاحت لنا التعامل مع الملفات بنوعيه الإدخال والإخراج .

بداية يجب أن نستدعي مكتبة الكلاسات لعمليات الإدخال والإخراج وهي مكتبة الـ IO ونستطيع إستدعائها و إستخدامها بالجملة Using المعروفة كالتالي :

```
using System.IO;
```

وإليك البرنامج التالي لنناقشه بالتفصيل .

قم بإنشاء ملف نصي على القرص C وسمه ما تريد وأي إمتداد تريد سأسميه هنا 1.txt وقم بكتابة الكود التالي :

```
using System;
using System.IO;

class Class1
{
	static void Main()
	{
		StreamReader r1 = new StreamReader("c:\\1.txt");
		string line1 ;
		while ( (line1 = r1.ReadLine()) != null )
			Console.WriteLine(line1);
		r1.Close();
	}
}
```

نلاحظ من البرنامج السابق :

- إستخدمنا مكتبة الإدخال والإخراج Input / Output لتنفيذ جمل الطباعة والقراءة من خلال ملف .
- أنشأنا كائن من نوع StreamReader وهو كائن للقراءة عن ملف موجود وأسندنا له مكان الملف وهنا يجب التنبيه إلى أنك يجب أن تكتب الإمتداد للملف بالكامل وذلك بإسم القرص ثم أتبعه برمز الـ (:) ثم أتبعه بشحطتين مائلتين (\\ ) ثم إسم الملف و إمتداده . وكل هذا يجب أن يكون بين علامتين تنصيص (" ") لأنه عبارة عن متغير نصي String .
- عرّفنا متغير نصي وذلك لوضع سطر فيه من الملف .
- قمنا بوضع حلقة تكرارية لأخذ سطر سطر من الملف من خلال المتغير r1 الذي يدل على عملية القراءة فقط وقمنا بوضع شرط التوقف عندما يصل الملف إلى النهاية أي عندما يكون المتغير line1 فارغاً . ويجب التنبيه هنا أن الفراغ لا يكون إلا في نهاية الملف فإذا قمنا بإضافة سطر فارغ بواسطة Enter فلا يعتبره فراغاً وإنما سطر فارغ ويقوم بطباعته .
- وفي كل حلقة نقوم بطباعة السطر بواسطة الكونسول Console .
- قمنا بإغلاق الملف ويجب علينا إغلاقه بعد إستعماله بكتابة إسم الكائن ثم إتباعه بكلمة Close والهدف من ذلك إزالته من الذاكرة أو حتى حمايته من العبث به والأخطاء .

كانت العملية السابقة للقراءة فقط ويجب إعداد ملف مسبقاً .

أما بالنسبة لعمليات الكتابة فإليك المثال التالي :

```
using System;
using System.IO;

class Class1
{
	static void Main()
	{
		StreamWriter w1 = new StreamWriter("c:\\2.txt");
		for ( int i=1 ; i<=5 ; i++ )
			w1.WriteLine(i);
		w1.Close();
	}
}
```

نلاحظ من البرنامج السابق :

- إستخدمنا مكتبة الإدخال والإخراج Input / Output لتنفيذ جمل الطباعة والقراءة من خلال ملف .
- أنشأنا كائن من نوع StreamWriter وهو كائن للكتابة في ملف غير موجود نقوم نحن بإنشاءه وبأي إمتداد نريد وأسندنا له مكان الملف الجديد وهنا يجب التنبيه إلى أنك يجب أن تكتب الإمتداد له بالكامل وذلك بإسم القرص ثم أتبعه برمز الـ (:) ثم أتبعه بشحطتين مائلتين (\\ ) ثم إسم الملف و إمتداده . وكل هذا يجب أن يكون بين علامتين تنصيص (" ") لأنه عبارة عن متغير نصي String . ويمكن إضافة كلمة True بعد إسم الملف كالتالي :
-

```
StreamWriter w1 = new StreamWriter("c:\\2.txt",true);
```

وتستخدم الجملة السابقة في حالة أنك رغبت في الإحتفاظ بالمحتويات القديمة للملف وتريد الإضافة فقط .

- قمنا بوضع حلقة تكرارية لإضافة سطر من الملف من خلال المتغير w1 الذي يدل على عملية الكتابة على الملف .
- وفي كل حلقة نقوم بطباعة السطر بواسطة الكونسول Console .
- قمنا بإغلاق الملف ويجب علينا إغلاقه بعد إستعماله بكتابة إسم الكائن ثم إتباعه بكلمة Close والهدف من ذلك إزالته من الذاكرة أو حتى حمايته من العبث به والأخطاء . وننبه أيضاً هنا أنك إذا لم تقم بإغلاقه فلن يعدل أي شيء عليه .

وفي كلا العمليتان السابقتان فبإمكانك الكتابة أو القراءة خانة خانة أو بسطر . فإذا أردت أن تكتب خانة خانة وعلى السطر يمكنك ذلك وبتعديل بسيط وهو :

```
w1.Write(i);
```

أما بالنسبة للقراءة :

```
using System;
using System.IO;

class Class1
{
    static void Main()
    {
		StreamReader r1 = new StreamReader("c:\\1.txt");
		char[] c = null;
		while (r1.Peek() >= 0)
		{
			c = new char[1];
		r1.Read(c, 0, c.Length);
		Console.Write(c);
	  }
    }
```

```
}
```

وجرب إذا جعلت طول المتغير الخاني c يساوي 5 فستلاحظ الفرق .

عملية التغليف Boxing :
تكلمنا سابقا ًعن الأنواع وتكلمنا أيضاً أنه يوجد نوع إسـمه Object وهـو يأخذ جميـع الأنـواع وإذا قمنـا بإسناد قيمة إليه تسمى هذه العملية Boxing وإذا قمنا بأخذ قيمة منه مـن خـلال متغيـر آخـر تـسمى العملية UnBoxing وإليك مثالاً على ذلك :

```
using System;

class Class1
{
    static void Main()
    {
            int x ;
            string s ="Welcome" ;

            object o ;
            o = 5 ; //Boxing
            x = (int)o ; // UnBoxing
            Console.WriteLine(x);

            o = "RTAQ" ; //Boxing
            s = (string)o ; //UnBoxing
            Console.WriteLine(s);
    }
}
```

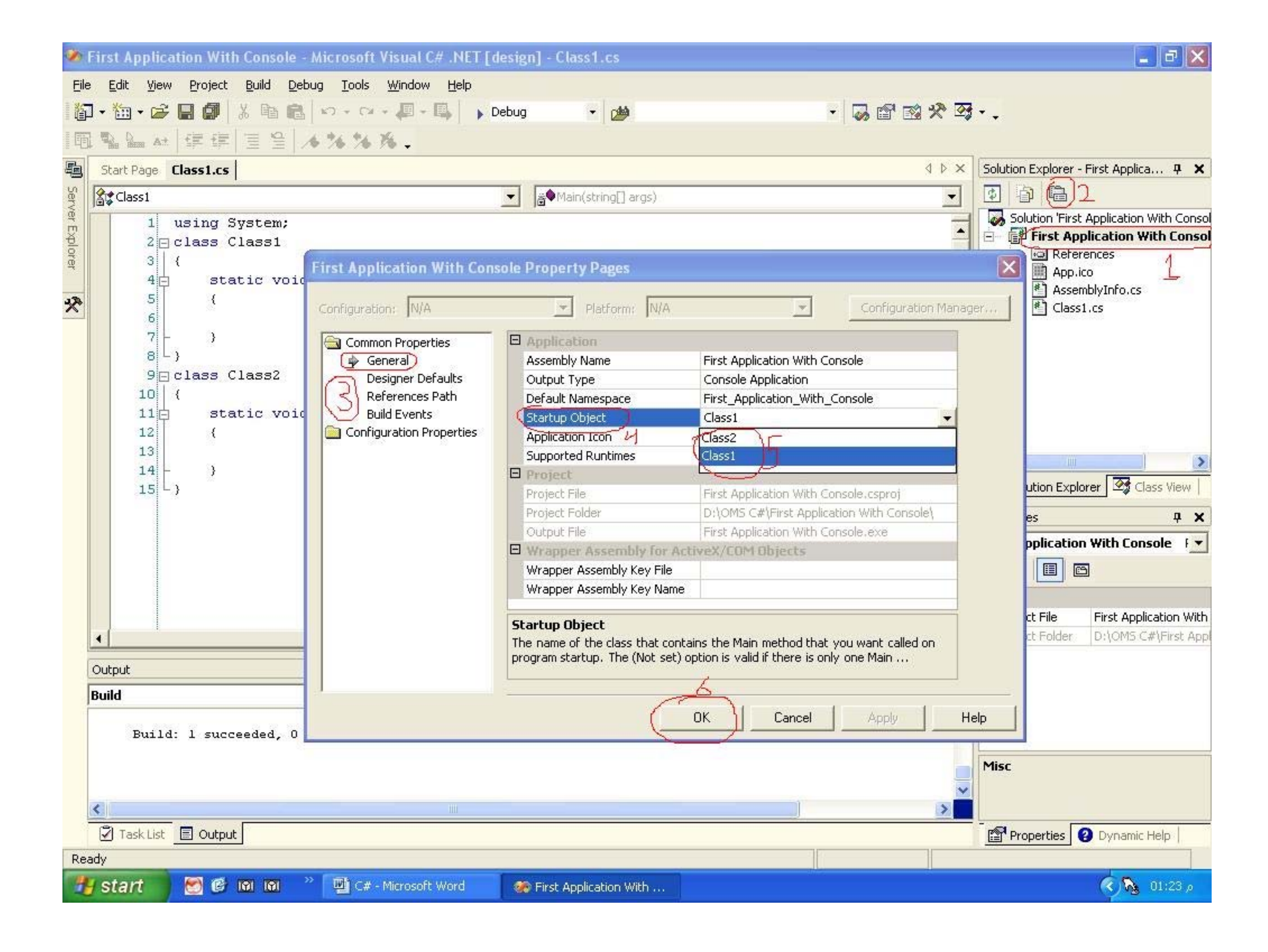
إستخدام دوال بواسطة الكلمة namespace :

بإمكانك تقسيم البرنامج إلى قسمين وبواسطة كلمة تجميع الـدوال إلـى مكتبـات namespace وإليـك مثالاً على ذلك :

```
using System;

namespace C1
{
    class Class1
      {
            static void Main()
            {
                  Console.WriteLine("I am Now In Class 1 ...");
                  C2.Class2.Fun();
            }
    }
}
namespace C2
{
      class Class2
      {
            public static void Fun()
            {
                  Console.WriteLine("I am Now In Class 2 ...");
            }
      }
}
```

ولا تنسى أن تضع نقطة البداية بواسطة الكلاس 1 كالتالي وكما قلنا سابقاً :



إختر مشروعك من خلال الرقم 1 ثم قم بإختيار الخصائص من خلال الرقم 2 أو من خلال الزر الأيمن للماوس ومن ثم إختر خصائص ، ثم قم بإختيار عام كما في الرقم 3 ثم إذهب إلى إقلاع الكائن كما في الرقم 4 ثم قم بإختيار أي كلاس من الكلاسات الموجودة عندك كما في الرقم 5 ثم قم بالضغط على موافق وهنا يقوم بالبدء من الكلاس الذي حددته .

ولكن هنا ستلاحظ وجود C1.Class1 فقط .

لحظ هنا أننا قمنا بإستدعاء الدالة الموجودة في مكتبة ثانية وذلك بذكر إسم المكتبة ثم إسم الدالة مباشرة ويجب أن تنتبه هنا أنه يجب أن تكون الدالة المستدعاة من نوع عام Public .

ويمكنك فعل السابق وبواسطة إستدعاء المكتبة بإستعمال الجملة التالي :

```
using C2;
```

ومناداة الكلاس مباشرة كالتالي :

```
Class2.Fun();
```

ليصبح البرنامج كالتالي :

```
using System;
using C2;
```

```
namespace C1
{
    class Class1
      {
            static void Main()
            {
                  Console.WriteLine("I am Now In Class 1 ...");
                  Class2.Fun();
            }
    }
}

namespace C2
{
      class Class2
      {
            public static void Fun()
            {
                  Console.WriteLine("I am Now In Class 2 ...");
            }
      }
}
```

قد أكون هنا قد أوصلت إليك فكرة إستخدام الملفات للعمليات المختلفة وأيضاً فكرة تقسيم البرنامج إلى أجزاء منفصلة وذلك لتخصيص المشكلة بعدة أسطر بدلاً من آلاف الأسطر .

O M S

بسم الله الرحمن الرحيم

المحاضرة الثانية عشر :

التوارث ومفهوم الـ Scope – قسم الـ Console :

لو قمنا بكتابة البرنامج التالي :

```
using System;

class Class1
{
	static void Main()
	{
		int x = 5 ;
		Console.WriteLine(x);
	}
	int x = 100 ;
}
```

فماذا تتوقع أن يحدث ؟؟ وما القيمة الناتجة من ذلك ؟؟

هنا سنتعرف على مفهوم Scope كالتالي :

- يعرف Scope بأنه النطاق الذي يتم تعريف المتغير به فلو لاحظنا في المثال السابق فنطاق المتغير x الموجود في الدالة الرئيسية هو :

```
	{
		int x = 5 ;
		Console.WriteLine(x);
	}
```

أي أنه بين قوسي البداية والنهاية . وهناك قوانين تحكمه فمن هذه القوانين أنه يرى في كل داخل الأقواس فقط أما خارجها فلا يرى أبداً وغالباً يتم إزالته من الذاكرة وحذف المكان الذي حذفه عند الوصول إلى قوس النهاية (}) . ومن هذه القوانين أيضاً أنه يشاهد في النطاق الداخل مالم يكون هناك متغير يحمل نفس الإسم وإن كان فسوف يعمل له Override .

ففي المثال السابق قمنا بتعريف المتغير x الخارجي وقمنا بإستخدام متغير آخر يحمل نفس الإسم في الجزء الداخلي وفي جملة الطباعة لم يتعرف المترجم سوى على المتغير الذي رآه آخر مرة وهو الداخلي .

لاحظ معي النتائج في المثال التالي :

```
using System;

class Class1
{
	int x = 43 ;
	static void Main()
	{
		Class1 C1 = new Class1();
		Console.WriteLine(C1.x);
		int x = 12 ;
		Console.WriteLine(x);
	}
}
```

ومن هنا يمكن القول أننا بإمكاننا أن نفتح أقواس بدون إسم دوال أو كلاسات كالتالي :

```
using System;

class Class1
{
	static void Main()
	{
		Console.Write("Welocme");
		{
			Console.Write(" In This");
			{
				Console.WriteLine(" Program");
```

```
                }
            }
        }
}
```

ونبه هنا أنه لا يجوز تعريف نفس متغير موجود في القوس الأعلى منه يحمل نفس الإسم .
كما في المثال التالي :

```
using System;

class Class1
{
        static void Main()
        {
            int x = 15 ;
            {
                int x = 5 ;
            }
        }
}
```

سينتج معك خطأً . ولكن إذا قمنا بتعريفه في سكوبين متساويين ففي هذه الحالة الناتج صحيح كما يلي :

```
using System;

class Class1
{
        static void Main()
        {
            {
                int x = 5 ;
            }
            {
                int x = 7 ;
            }
        }
}
```

سنتحدث الآن عن موضوع جديد وهو الـ Constructor :
هل فكرت مرة من المرات لماذا تقوم بكتابة object من الكلاس مثلاً على الصيغة العامة ؟

```
Class1 C1 = new Class1();
------------------------------------^
```

الكائن الجديد هو من نوع الكلاس الأب وعملية فتح الأقواس هنا تعني أنه قم بإستدعاء الكونستركتر .
ولكن ما هو الكونستركتر ؟
هو عبارة عن دالة أو ميثود تحمل نفس إسم الكلاس (لا يجوز إستخدامها إلا مع الكلاسات فقط) ولا ترجع أي قيمة وليس لها أي نوع .
وإليك مثال عليها :

```
using System;

class Class1
{
        public Class1()
        {
            Console.WriteLine("Welcome To RTAQ");
        }
        static void Main()
        {

        }
}
```

ولكن هنا السؤال المهم : ما وظيفته ؟

يعتبر الكونستركتر الميثود الإفتتاحي أي أنك عندما تقوم بإنشاء أي كائن من نوع الكلاس الذي يحتوى على كونستركتور سينفذ كل الجمل الموجودة في داخله .
فلو نفذنا المثال التالي :

```
using System;

class Class1
{
	static void Main()
	{
		Class2 C2 = new Class2();
	}
}

class Class2
{
	public Class2()
	{
		Console.WriteLine("Welcome");
		Console.WriteLine("In");
		Console.WriteLine("C#");
	}
}
```

في المثال السابق لم نقوم سوى بإنشاء كائن فقط من نوع Class2 والباقي يقوم به الكونستركتر .
وبإمكانك أيضاً وضع باراميتر أو وسائط للكونستركتر (وهي المهمة الرئيسية له) كما في المثال التالي :

```
using System;

class Class1
{
	static void Main()
	{
		Class2 C2 = new Class2(1,3);
	}
}

class Class2
{
	public Class2(int x,int y)
	{
		Console.WriteLine("The Sum Is : {0}",x+y);
	}
}
```

وننبه هنا أنه يجب عليك بعث قيم عندما تقوم بإنشاء كائن وغير ذلك سينتج خطأ في البرنامج .
وأيضاً يجب وضع كلمة public لكي نستطيع إستدعاءه من كلاس آخر كما تعلمت سابقاً .

حسناً ولكننا كنا سابقاً نستدعي الكلاس بدون كتابة الكونستركتر فكيف ذلك ؟
تقوم لغة السي شارب بوضع كونستركتور افتراضي ووهمي وفيه التالي :

```
	public Class1()
	{
	}
```

حيث أن الإسم Class1 هو إسم الكلاس .

التوارث Inheritance :

فكرة التوارث أهم فكرة من أفكار البرمجة الموجهة بالأهداف OOP ونستطيع شرح هذه الفكرة بتطبيق مبدأ الأب و الإبن . فهنا يقوم الإبن بتوارث جميع الميثود الموجودة في الأب وأيضاً يستطيع حمل خصائص لا يحملها الأب .
وهنا التوارث لا يحدث إلا لكلاسين فقط . وتتم هذه الطريقة بكتابة عملية التوارث في الكلاس الإبن بوضع إشارة (:) بعد الكلاس وكتابة الكلاس الأب كالتالي :

```
using System;

class Class1:Class2
{
	static void Main()
	{
		Class1 C1 = new Class1();
		C1.print();
	}
}

class Class2
{
	public void print()
	{
		Console.WriteLine("Welcome In C#");
	}
}
```

لاحظ في المثال السابق أننا قمنا بإستعمال دالة `print()` **وهي غير موجودة في الكلاس** `Class1`
ولكن هنا قمنا بعملية وراثة من كلاس أخرى .
وعملية التوارث تتم فقط على الدوال ذات النوع `public` **وكل الباقي يعتبر خاص** `Private` **.**

ولكن ماذا لو إحتوى الكلاس الأب كونستركتور كما في المثال التالي ؟

```
using System;

class Class1:Class2
{
	static void Main()
	{
		Class1 C1 = new Class1();
	}
}

class Class2
{
	public Class2()
	{
		Console.WriteLine("Welcome In C#");
	}
}
```

أكيد سيعمل على تنفيذ الكونستركتر للأب .
سنناقش الكود التالي لتتضح الصورة لك أكثر :

```
using System;

class Class1:Class2
{
	public Class1()
	{
		Console.WriteLine("I am Now In Class 1");
	}
	static void Main()
	{
		Class1 C1 = new Class1();
	}
}
```

```
class Class2:Class3
{
	public Class2()
	{
		Console.WriteLine("I am Now In Class 2");
	}
}

class Class3
{
	public Class3()
	{
		Console.WriteLine("I am Now In Class 3");
	}
}
```

لاحظ هنا أننا نقوم بإنشاء كائن من نـوع Class1 **لكـن هـذا الكـلاس هـو عبـارة عـن وارث مـن الكـلاس** Class2 **ولهذا فإنه سينفذ الكونستركتر الموجود في الكلاس الأب ولكن الكلاس الأب عبـارة عـن وارث من الكلاس** Class3 **لذلك سيقوم بتنفيذ الكونستركتر الموجود في الكلاس** Class3 **.**
وهنا سيكون الناتج كالتالي :

```
I am Now In Class 3
I am Now In Class 2
I am Now In Class 1
```

ونهايةً ماذا لو حمل الإبن دالة موجودة في الأب كما في المثال التالي ؟

```
using System;

class Class1:Class2
{
	public void print()
	{
		Console.WriteLine("I am Now In Class 1");
	}
	static void Main()
	{
		Class1 C1 = new Class1();
		C1.print();
	}
}

class Class2
{
	public void print()
	{
		Console.WriteLine("I am Now In Class 2");
	}
}
```

فماذا تتوقع ؟
هنا سيقوم بحذف الميثود الموجودة في الكلاس الأب وسيقوم بتنفيـذ الميثـود الموجـود فـي الكـلاس الإبن . وسيكون الناتج هنا :

```
I am Now In Class 1
```

O M S

بسم الله الرحمن الرحيم

المحاضرة الثالثة عشر :

الـ Interface وعملية توارث الأب من الإبن – قسم الـ Console :

تعلمنا سابقاً كيف نقوم بتوريث كلاس لبعض الميثود والدوال الموجودة في كلاس آخر وسمينا هذه العلاقة بعملية توارث الإبن من خصائص أبيه .

فلو نظرنا إلى المثال التالي :

```
using System;

class Class1
{
	static void Main()
	{
		B b = new B();
		b.PrintA();
		b.PrintB();
	}
}

class A
{
	public  void PrintA()
	{
		Console.WriteLine("Class A");
	}
}
class B:A
{
	public void PrintB()
	{
		Console.WriteLine("Class B");
	}
}
```

لاحظنا أننا عرفنا كلاس إبن من كلاس أب وورث جميع صفاته بالإضافة إلى الصفات الموجودة عنده .
ولكن هل يجوز لنا أن نأخذ كائن من نوع الأب يحتوي على خصائص إبنه ؟

لكي تتضح لك الصورة أكثر أنظر إلى المثال التالي :

```
using System;

class Class1
{
	static void Main()
	{
		B b = new B();
		b.Print();
	}
}

class A
{
	public void Print()
	{
		Console.WriteLine("Class A");
	}
}
```

```
class B:A
{
	public void Print()
	{
		Console.WriteLine("Class B");
	}
}
```

عند تنفيذ الجملة التالية `b.Print();` ماذا تتوقع أن يطبع ؟

أكيد سيقوم بطباعة الدالة الموجودة في A لأنه الأب !! . وهذا كلام خطأ فعندما يصل إلى الجملة المعنية يأخذ دالة الأب بالحسبان ويخزنها عنده وعندما يصل إلى دالة الإبن يقوم بعمل Override للدالة السابقة (التابعة للأب) ويخزن بدلاً منها دالة الإبن وبالتالي سيقوم بطباعة الجملة التالية :

```
Class B
```

حسناً إليك طريقة توريث الأب الصفات والخصائص من إبنه بالكود التالي :

```
using System;

class Class1
{
	static void Main()
	{
		A a = new B();

		a.Print();
	}
}

class A
{
	public virtual void Print()
	{
	}
}
class B:A
{
	public override void Print()
	{
		Console.WriteLine("Class B");
	}
}
```

ونستطيع ملاحظة التالي :

- قمنا بعمل كائن للأب من نوع الإبن كما في الجملة التالية :

```
A a = new B();
```

- يجب وضع دالة من نفس نوع و إسم الدالة الموجودة في الإبن عند الأب كما في الجملتان المنفصلتان :

```
public virtual void Print()
public override void Print()
```

- قمنا بإضافة كلمة `virtual` في الدالة الموجودة في كلاس الأب والتي تخبرنا بأخذ المحتوى للدالة من الإبن .

- قمنا بإضافة كلمة `override` في الدالة الموجودة في كلاس الإبن والتي تخبرنا بأننا سنقوم بالكتابة فوق دالة من نفس الإسم والنوع عند كلاس الأب .

- لاحظ أننا قمنا بكتابة إسم الدالة فقط بدون كتابة الكود داخلها لأننا سندخله لاحقاً .

والآن لعلك تتساءل : ولكن ما الفائدة من هذه العملية ؟؟

سأضرب لك مثالاً على كيفية الإستفادة منه بالمجال العملي :
لنفرض أننا في معرض سيارات وأردت معرفة سرعة سيارة معينة . فلنفرض هنا أنه يوجد كلاس أساسي وهو السيارة ويوجد كلاسات تتوارث صفات السيارة وبها تخصيصات مثل أنواع السيارات المختلفة : مرسيدس ، تويوتا ، هوندا ، وهونداي ، وغيرها ... فهنا يجب أن تعرف نوع السيارة ولنفترض أنها من نوع هوندا سيفيك (CIVIC) ولكي نقوم بعرض سرعة السيارة يجب معرفة صنفها ونوعها ، وكل الكلام السابق يمكن ترجمته إلى كود كالتالي :

```
using System;

class CarExhibition
{
	static void Main()
	{
		Honda h1 = new CIVIC();

		h1.Speed();
	}
}

class Honda
{
	public virtual void Speed()
	{
	}
}
class CIVIC:Honda
{
	int Spd = 160 ;
	public override void Speed()
	{
		Console.WriteLine("Honda CIVIC Speed Is : {0} MPH",Spd);
	}
}
```

ولننتقل إلى موضوع جديد وهو الواجهات Interface :
قد يذهب ذهنك إلى الشاشات و الواجهات المستخدمة في تطبيقات الويندوز !! ولكن الموضوع مختلف هنا . فيمكن تعريف الـ Interface كالتالي :
هو عبارة عن وحدة نمطية (مثل الكلاس) يحتوي على أسماء وأنواع لدوال و ميثود ولكن من غير كود ، حيث يتم تعريف كود في كلاس آخر يرثها جميعاً ويقوم بتعريفها ، ولا يحتوي على بارامتير أي أنه لا يستعمل الأقواس بعد تعريفه ( ) ، وإليك الصيغة العامة له :

```
interface InterfaceName
{
	Methods();
}
```

ولنأخذ مثالاً على كيفية إستعماله :

```
using System;

class Class1
{
	static void Main()
	{
		Class2 c1 = new Class2();
		Console.Write("Enter First Number : ");
		float x = float.Parse(Console.ReadLine());
```

```
            Console.Write("Enter Second Number : ");
            float y = float.Parse(Console.ReadLine());

            c1.Sum(x,y);
            c1.Subtraction(x,y);
            c1.Product(x,y);
            c1.Divide(x,y);
        }
}

interface Operation
{
        void Sum(float x , float y);
        void Subtraction(float x , float y);
        void Product(float x , float y);
        void Divide(float x , float y);
}
class Class2:Operation
{
        public void Sum(float x , float y)
        {
            Console.WriteLine("{0} + {1} = {2}",x,y,x+y);
        }
        public void Subtraction(float x , float y)
        {
            Console.WriteLine("{0} - {1} = {2}",x,y,x-y);
        }
        public void Product(float x , float y)
        {
            Console.WriteLine("{0} * {1} = {2}",x,y,x*y);
        }
        public void Divide(float x , float y)
        {
            Console.WriteLine("{0} / {1} = {2}",x,y,x/y);
        }
}
```

يمكن تلخيص البرنامج السابق بالتالي :
- قمنا بإنشاء interface وسميناه Operation وقمنا بكتابة أربع دوال داخله .
- لاحظ أننا لم نقوم بكتابة كلمة public الدالة على إستعمال الدوال خارج الـ interface وأيضاً لم نقوم بكتابة الكود فيهم لأننا سنقوم بكتابتهم في كلاس آخر .
- عرفنا كلاس آخر وقمنا بتوريثه جميع الدوال الموجودة في الـ interface .
- قمنا بتحديد نوع الدوال داخل الدالة من خلال كلمة public لكي نستعملها في دالة أخرى .

من المثال السابق نكون قد فهمنا معنى الـ interface وكيفية إستعماله .
والفائدة منه هو أننا نقوم بإدراج أسماء الدوال التي يتطلبها المشروع ومن ثم نقوم بكتابة الكود الخاص بهم على مهلنا وفي كلاس آخر من حيث عملية التوارث .

O M S

بسم الله الرحمن الرحيم

المحاضرة الرابعة عشر :

الـ Sealed كلاس والـ Abstract كلاس – قسم الـ Console :

تعلمنا سابقاً كيفية التوارث بين الكلاسات المختلفة وأخذنا مفهوم الـ Interface وطريقة توريثها .
ولكن ؟!! ماذا لو أنك تريد كلاس يرث ولا يورِّث أي أنه عقيم !!!
في هذه الحالة نستخدم أو نضع كلمة Sealed قبل إسم الكلاس الذي لا نريد أن نورثه ، وإليك كيفية إستعماله والتعامل معه :

```
using System;

sealed class Class1
{
	static void Main()
	{
		Class2 c2 = new Class2();
	}
	public static void print()
	{
		Console.WriteLine("Welcome ...");
	}
}

class Class2:Class1
{
	public Class2()
	{
		//Constructor
	}
}
```

لاحظ أننا قمنا بوضع الكلاس 1 بحيث أنه لا يورِّث ولذلك إذا قمنا بتنفيذ البرنامج السابق ستظهر لنا الجملة التالي :

```
'Class2' :cannot inherit from sealed class 'Class1'
```

أي أن الكلاس الثاني لا يمكن أن يرث من الكلاس الأول الذي هو في الأصل عقيم .
وله تطبيقات كثيرة أي أنه يوجد لدينا العديد من الكلاسات التي لا يمكن أن نرث منها وعلى سبيل المثال الكلاس الخاص بالعمليات الحسابية الرياضية Math فلو أننا قمنا بتنفيذ الكود التالي :

```
using System;

class Class1:Math
{
	static void Main()
	{
		Console.WriteLine("Welcome ...");
	}
}
```

فإنه سينتج عندنا الخطأ التالي :

```
'Class1' : cannot inherit from sealed class 'System.Math'
```

وأهم إستعمال له أننا قمنا بعمل كلاس لا نريد أن يورث منه أي ميثود أو أي خاصية داخله .

الـ Abstract ميثود و الـ Abstract كلاس :
تحدثنا سابقاً عن مفهوم الـ Interface وقنا أنه مشابه للكلاس وأنه يحتوي على ميثود ولا تحتوي على Implementation لها أي أننا لم نكتب محتوياتها ، ولكن ماذا لو كنا في كلاس وقمنا بإستعمال ميثود ولكننا نريد كتابة محتوياتها في كلاس آخر ، بمعنى أنه يوجد لدينا ميثود أو دالة لها تعريف فقط ولا تحتوي على كود فكيف نفعل ذلك ؟؟
هنا لا بد من جعل نوع الميثود Abstract وبالتالي أيضاً نوع الكلاس سيصبح Abstract ولكي تفهم على ما أقول إليك المثال التالي :

```
using System;

abstract class Class1
{
	static void Main()
	{
		Class2 c2 = new Class2();
		c2.Print();
	}
	abstract public void Print();
}

class Class2:Class1
{
	public override void Print()
	{
		Console.WriteLine("Test Abstract ...");
	}
}
```

في المثال السابق قمنا بتعريف دالة في Class1 وهي الدالة Print وقمنا بكتابة الكود لها في الكلاس الثاني .
وهنا يجب أن ننبه أنه ما دام عندنا ميثود أو دالة واحدة فقط نوعها Abstract فيجب أن نضع نوع الكلاس الحاوي لها من نفس نوعها أي Abstract ويمكن مشاهدة ذلك في المثال السابق .

جملتي الـ Set و الـ Get :
لو نظرنا إلى المثال التالي :

```
using System;

class Class1
{
	static void Main()
	{
		int x = 5 ;
		Console.WriteLine( x );
	}
}
```

ما هو الناتج ؟؟ أكيد الرقم 5 .
حسنا سنزيدها صعوبة ، إليك المثال التالي :

```
using System;

class Class1
{
	static void Main()
	{
		Class2 c2 = new Class2();
		Console.WriteLine( c2.x );
	}
}

class Class2
```

```
{
      int x = 5 ;
}
```

ما ناتجها ؟؟
قم بتنفيذها وسوف تظهر لك الجملة التالية في شاشة الـ Task List :

```
'Class2.x' is inaccessible due to its protection level
```

ولكن لماذا ؟؟
في البرنامج السابق إعتبر أن المتغير x هو متغير خاص بالكلاس الثاني وأننا لا نستطيع الحصول عليه ، ولكن ماذا لو أنني إضطررت لإستعماله ، أي أنني أريد طباعة قيمته أو حتى تعديله ، وذلك مع المحافظة على كونه خاص بالكلاس الثاني ؟؟
من هذا السؤال أتت جملتا Set و Get واللتان تعنيان أننا سنقوم بأخذ محتوى المتغيرات الخاصة وجعلها عامة وإليك مثالاً عليهما :

```
using System;

class Class1
{
      static void Main()
      {
            Class2 c2 = new Class2();
            Console.WriteLine( c2.SGXV );
            c2.SGXV = 7 ;
            Console.WriteLine( c2.SGXV );
      }
}

class Class2
{
      int x = 3 ;
      public int SGXV // Set & Get X Variable
      {
            set
            {     x = value ; }
            get
            {     return x ;  }
      }
}
```

يمكن ملاحظة التالي من المثال السابق :

- قمنا بكتابة دالة جديدة لها معنى (إختصار للدلالة على المعنى) .
- قمنا بكتابة جملة Set المحجوزة والتي تدل على عملية إعطاء قيمة .
- قمنا بكتابة أو بإسناد قيمة إلى المتغير x الذي يعتبر متغير عالي ويمكن مشاهدته .
- قمنا بكتابة جملة Get المحجوزة والتي تدل على عملية إرجاع قيمة .
- قمنا بإرجاء قيمة المتغير x بإستخدام كلمة الإرجاع Return .

ولكن لعلك تسأل ما هي الكلمة المحجوزة value ولماذا لم تذكرها ؟؟
هذه كلمة تدل على قيمة تسند إلى المتغير من خلال دالة .

وبواسطة الجملتان Set و Get يمكن أن نخرج بمصطلحين للمتغير الخاص بواسطتهما وهما :
مصطلح الـ Read Only : وهنا نقوم بإستخدام جملة Get فقط ولا نستخدم جملة Set .
مصطلح الـ Write Only : وهنا نقوم بإستخدام جملة Set فقط ولا نستخدم جملة Get .

O M S

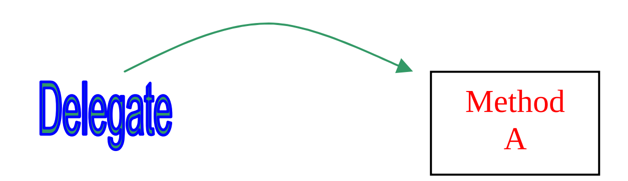
بسم الله الرحمن الرحيم

المحاضرة الخامسة عشر :

الـ Delegate والأحداث Event والـ Indexer – قسم الـ Console :

الـ Delegate (المندوب) :

يعرف الـ Delegate بأنه متغير يدل على دالة أو ميثود ويجب أن يكون نوعه من نفس نـوع الدالـة التـي يقوم بالتأشير عليها وإليك ما قلت :



وبالنسبة لتعريفه والتعامل معه إليك المثال التالي :

```
using System;

delegate void MyDelegate(int i);

class Class1
{
	public static void Main()
	{
		MyDelegate d1 = new MyDelegate(MethodA);
		MethodB(d1);
	}

	public static void MethodB(MyDelegate md)
	{
		md(21); //A الميثود إستدعينا كأننا
	}

	public static void MethodA(int i)
	{
		System.Console.WriteLine("Called by delegate : {0}.", i);
	}
}
```

في المثال السابق :

- **قمنا بتعريف Delegate من نوع الدالة التي سنقوم بالتأشير عليها .**
- **في الدالة الرئيسية قمنا بتعريف متغير من نوع Delegate وهو هنا d1 .**
- **ولاحظ هنا أننا قمنا بالتأشير على دالة من نفس نوع الـ Delegate .**
- **قمنا بإرسال المتغير إلى دالة تتعامل مع الـ Delegate بواسطة البارامتير .**
- **قمنا بإستدعاء المتغير من خلال البارامتير في الميثود .**
- **وهنا نفذها كأننا إستدعينا الدالة الرئيسية**

يمكنك أيضاً طالما أن النوع موحد في جميع الدوال أن تقوم بالتأشير على أكثر من ميثود في البرنـامج فعلى سبيل المثال ماذا لو أردنا أن نجري العمليات الأربعة بواسطة متغير واحد فقط ؟

إليك الحل في المثال التالي :

```
using System;

public delegate void MyDelegate(float x , float y);
class Class1
{
	static void Main()
	{
		Class2 c1 = new Class2();
		Console.Write("Enter First Number : ");
		float x = float.Parse(Console.ReadLine());
		Console.Write("Enter Second Number : ");
		float y = float.Parse(Console.ReadLine());

		MyDelegate d1 = new MyDelegate(c1.Sum); // add Sum
		d1 += new MyDelegate(c1.Subtraction); // Sum + Sub
		d1 += new MyDelegate(c1.Product); // Sum + Sub + Prod
		d1 += new MyDelegate(c1.Divide); // Sum + Sub + Prod + Div

		d1(x,y); // Call Four Method --> Sum + Sub + Prod + Div
	}
}

class Class2
{
	public void Sum(float x , float y)
	{
		Console.WriteLine("{0} + {1} = {2}",x,y,x+y);
	}
	public void Subtraction(float x , float y)
	{
		Console.WriteLine("{0} - {1} = {2}",x,y,x-y);
	}
	public void Product(float x , float y)
	{
		Console.WriteLine("{0} * {1} = {2}",x,y,x*y);
	}
	public void Divide(float x , float y)
	{
		Console.WriteLine("{0} / {1} = {2}",x,y,x/y);
	}
}
```

فكما لاحظنا في المثال السابق قمنا بإستدعاء أربعة دوال في جملة واحدة ، وأكثر فائدة لهذه الجملة في تطبيقات الويندوز التي سنتعرف عليها قريباً .

الأحداث Event :

والحدث غالباً يستدعى عندما نقوم بعمل شيء معين ، فمثلاً عندما نضع ناراً على ورقة يحدث حريق أي أننا قمنا بتعريف جزأين وهما سبب الحدث و الناتج من الحدث .
وهنا يتعامل الحدث مع مجموعة من الدوال بل بالأحرى يتعامل مع مجموعة من الـ Delegate وكتوضيح على مفهوم الحدث إليك المثال التالي :

```
using System;

public delegate void Paper();    // delegate declaration

class Class1
{
	static event Paper FireEvent; // event declaration
```

```
        static public void Main ()
        {
                FireEvent += new Paper(Burn);
                FireEvent();
        }

        static private void Burn()
        {
                Console.WriteLine("The Event Of ( Fire + Paper ) Is : Burn");
        }
}
```

وغالباً نستعمله في أحداث الأدوات ، فمثلاً في الزر أو أي كبسة في أي برنامج هناك حدث النقر مرة واحدة أو النقر مرتين أو مرور الفأرة فوقه .... الخ .

وننبه هنا أن الحدث لا يأخذ سوى مؤشرات للدوال أي Delegate .

وإليك مثالاً آخر على مفهوم الحدث :

```
using System;
public delegate void MyDelegate();    // delegate declaration

public interface I
{
        event MyDelegate MyEvent;
        void FireAway();
}

public class MyClass: I
{
        public event MyDelegate MyEvent;

        public void FireAway()
        {
                if (MyEvent != null)
                        MyEvent();
        }
}

public class Class1
{
        static private void f()
        {
                Console.WriteLine("This is called when the event fires.");
        }

        static public void Main ()
        {
                I i = new MyClass();

                i.MyEvent += new MyDelegate(f);
                i.FireAway();
        }
}
```

التعامل مع المصفوفات الجزئية بواسطة الـ Indexer :
تعرفنا على مفهوم المصفوفات قبلاً ، لو نظرنا إلى المثال التالي :

```
using System;

class Class1
{
	static public void Main ()
	{
		int []a = new int[20];
		Random r1 = new Random();
		for ( int i = 0 ; i < 20 ; i++)
		{
			a[i] = r1.Next(20);
			Console.WriteLine("a[{0}] = {1}",i,a[i]);
		}
	}
}
```

لعرفنا أن المثال يتحدث عن إعطاء قيم أولية للعناصر في المصفوفة عشوائياً كما شرحنا عنها سابقاً . ولكن لو كانت المصفوفة في كلاس آخر هل نستطيع التعامل معها أي كما في المثال التالي :

```
using System;

class Class1
{
	static public void Main ()
	{
		Class2 c2 = new Class2();
		c2.a[15] = 7 ;
		Console.WriteLine(c2.a[3]);
	}
}
class Class2
{
	int []a = new int[20];
	public Class2 ()
	{
		Random r1 = new Random();
		for ( int i = 0 ; i < 20 ; i++)
		{
			a[i] = r1.Next(20);
		}
	}
}
```

كما في المثال السابق سينتج معنا خطأين من نوع :

```
'Class2.a' is inaccessible due to its protection level
```

ولكن كيف أستطيع التعامل معها على الرغم من أنها خاصة بالكلاس الثاني ؟؟

سبق لنا أن وقعنا في هذه المشكلة أتذكر (بإستعمال جملتي Set و Get) وإليك الحل لهذه المشكلة كما في المثال التالي :

```
using System;

class Class1
{
	static public void Main ()
	{
		Class2 c2 = new Class2();
		c2[18] = 7 ;
```

```
            Console.WriteLine(c2[18]);
        }
}
class Class2
{
        int []a = new int[20];
        public Class2 ()
        {
                Random r1 = new Random();
                for ( int i = 0 ; i < 20 ; i++)
                {
                        a[i] = r1.Next(20);
                }
        }
        public int this [int index1]
        {
                set
                {
                        if ( index1 >= 0 && index1 <20)
                                a[index1] = value ;
                }
                get
                {
                        if ( index1 >= 0 && index1 <20)
                                return a[index1];
                        else
                                return 0;
                }
        }
}
```

لاحظ معي :

- قمنا بإنشاء دالة بإستخدام الجملة `public int this [int index1]` في الدالة الثانية .
- ذكرنا فيها الكلمة المحجوزة This الدالة على الكلاس الثاني ثم عرفنا الـ Indexer .
- عرفنا المتغير index1 كـ Index نتعامل معه في الكود التالي .
- قمنا بإستعمال جملتي Set و Get .
- قمنا بإدخال شروط في جملة Set .
- أما في جملة Set فيجب أن نضع فيها جملة else ومعناها أنه إذا لم يكن العنصر موجود .

وهذا مثال آخر على الـ Indexer :

```
using System;
class IndexerClass
{
        private int [] myArray = new int[100];
        public int this [int index]   // Indexer declaration
        {
                get
                {
                        // Check the index limits.
                        if (index < 0 || index >= 100)
                                return 0;
                        else
                                return myArray[index];
                }
                set
                {
                        if (!(index < 0 || index >= 100))
                                myArray[index] = value;
```

```
			}
		}
}

public class Class1
{
	public static void Main()
	{
		IndexerClass b = new IndexerClass();
		// Call the indexer to initialize the elements #3 and #5.
		b[3] = 256;
		b[5] = 1024;
		for (int i=0; i<=10; i++)
		{
			Console.WriteLine("Element #{0} = {1}", i, b[i]);
		}
	}
}
```

خاتمة :

إلى هنا نكون قد ختمنا العمل مع الكونسول Console و سنبدأ من الدرس التالي بإستخدام تطبيقات النوافذ Windows Application .

O M S

Omssd84@hotmail.com

بسم الله الرحمن الرحيم

المحاضرة السادسة عشر :

بيئة الفيجوال ستوديو دوت نيت – قسم الـ Windows :

- قبل التحدث والدخول إلى هذه البيئة يجب أن تمتلك نسخه منها وعنوانها :
Microsoft Visual Studio .NET 2003 وهي على الأغلب من 6 سيديات متكونة من :

2 سي دي للدوت نيت 2003
3 سي دي للمكتبة MSDN
1 سي دي Component وهو عبارة عن مكونات ما قبل البداية
1 سي ي SQL 2000 Server ((إضافي)) للتعامل مع قواعد البيانات.

لنبدأ الآن :
قم بتتبع التسلسل التالي (حسب المسار الذي إخترته في عملية التنزيل) :
Start → Program File → Microsoft Visual Studio .NET 2003
→ Microsoft Visual Studio .NET 2003

بعد أن تتبعت المسار ستظهر لك النافذة الرئيسية كالتالي :

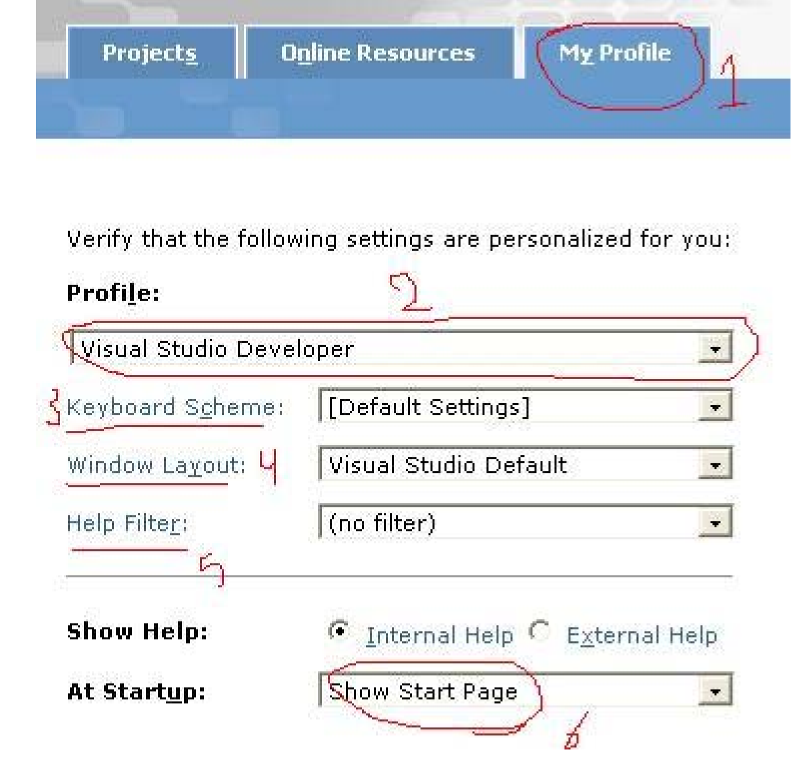
لاحظ معي وبالترتيب لكي تتعرف على واجهة البرنامج :

1 – وهو شريط العنوان والكل يعرفه .
2 – شريط القوائم و به أوامر للمساعدة في بناء المشروع في بيئة الدوت نيت
3 – شريط الإختصارات و به أوامر مختصرة ((كثيرة الإستعمال)) من شريط القوائم ((2))
4 – صفحة البداية وهي الصفحة الرئيسية لهذه البيئة وتتكون من :

- المشاريع Projects ((6)) وتتكون من التالي :
- نافذة المشاريع وتحتوي على آخر أربع مشاريع ((رقم 7)) قمت بالعمل بهم مع التاريخ حيث يكتب التاريخ إما اليوم أو الأمس أو تاريخ العمل به ((التعديل عليه)) .
- أزرار إنشاء أو فتح مشاريع سابقة ((12)) ((13)) على التوالي .

- المصادر من الإنترنت Online Resource :
والمهمة لهذه الصفحة الحصول على الأمثلة أو طرح المشاكل التي تواجهك في منتديات البرمجة لموقع مايكروسوفت للحلول المباشرة وهذه الخدمة متوفرة شريطة الربط مع الإنترنت .

- ملفك الشخصي My Profile :
هنا يمكنك التعديل على الواجهة بشكل عام أو إختصارات لوحة المفاتيح كالتالي :
لو فتحت هذه النافذة لوجدت الشاشة كالتالي :



((3)) يمكنك إختيار أي إختصارات للغة قد تعودت عليها من قبل كالسي ++ 6
((4)) يمكنك تغيير الشكل العام أو الإطار العام لبيئة الدوت نيت
((5)) يمكنك أن تختار طريقة عرض المساعدة للغة قد تعودت عليها من قبل
* كل الأرقام السابقة يمكن أن تعطيك ملف خاص كما في ((2))
و الإختيار السادس هو إجراء حدث عند بداية تنفيذ بيئة الدوت نيت .

5 - شريط لعرض النوافذ الموجودة أمامك فإذا إختفت واحدة ما عليك سوى الضغط على إسمها وستظهر لك حالاً .

9 – شريط الأدوات ولكن في السي شارب Console لن نستعمله فهو خاص بتطبيقات النوافذ .

10 – نافذة ملفاتك في مشروعك الذي تعمل عليه الآن .

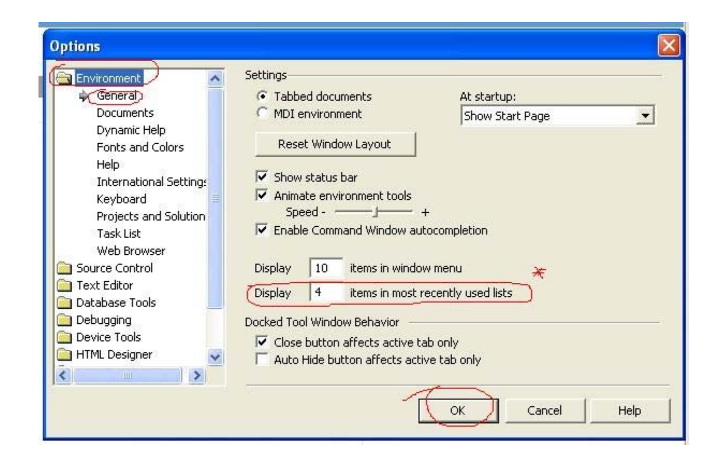
11 – نافذة المساعدة .

14 – في كل نافذة ستجد إشارة الدبوس هذه ومعناها أخفي تلقائياً بمعنى أنه إذا ذهبت الماوس من فوق تلك النافذة قم بإخفائها تلقائية . وإذا إقتربت منها قم بإظهارها تلقائياً .

ملاحظة (1) :
في القسم رقم ((6)) من الصورة السابقة يمكنك زيادة عدد المشاريع عن أربعة وذلك بالمسار التالي :

Tools → Option → Environment → General → Display recently

كما في الصورة التالية :

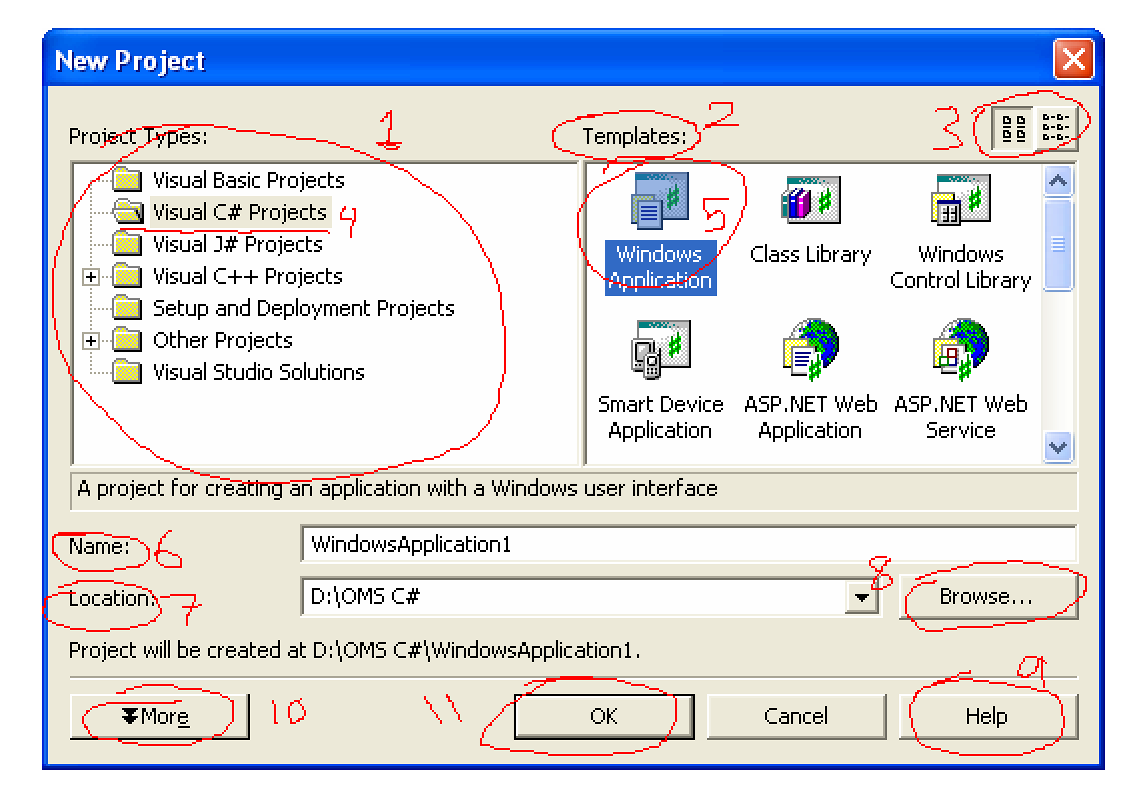


فقم بزيادة العدد عند إشارة النجمة ((*)) ثم قم بالنقر على موافق .

ملاحظة (2) :
إذا بدأت البيئة و لم تظهر لك نافذة البداية ((صفحة البداية)) إذهب إلى :
Help → Show Start Page

الآن وبعد ما تعرفت على أقسام النافذة الرئيسية لنبدأ بمثالنا الأول بالـ Windows Application :

إذهب الآن إلى File → New → Project .
أو قم بالضغط على Ctrl + Shift + N .
أو قم بالنقر على زر المشروع الجديد New Project من شريط الإختصارات .
إذا نجحت في ذلك ستظهر لك النافذة التالية :



لنقف قليلاً عند هذه النافذة لكي نبين أجزائها :

1 – أنظر هنا في القسم ((1)) ستلاحظ أسماء جميع اللغات المستعملة في بيئة الدوت نيت وبهذا تكون شركة مايكروسوفت قد نجحت في تجميع جميع اللغات في إطار واحد .

2 – تجد جميع الأقسام ((القوالب)) التي يمكنك تصميمها مثل تطبيقات الويندوز وتطبيقات الكونسول وتطبيقات السمارت ديفايس ((Pocket PC)) وذلك في القسم ((2)) .

3 – يمكنك أن تصغر وتكبر الأيقونات الموجودة في القسم ((2)) بواسطة الأزرار في القسم ((3)) .

4 – إذن الآن لنقم بفتح مشروع سي شارب ، وذلك بالضغط على ((4)) .

5 – ثم بعد ذلك بالضغط على ((5)) ولاحظ أنها تطبيقات لـ Windows ، وهي أي برنامج سبق لك أن إستعملته يحتوي على أزرار أو صناديق نصوص أو ... إلخ .

6 – يمكنك تسمية المشروع من الرقم ((6)) وتذكّر أنه سيقوم بتسمية الملفات كلها بهذا الإسم ((ملفات العمل)) .

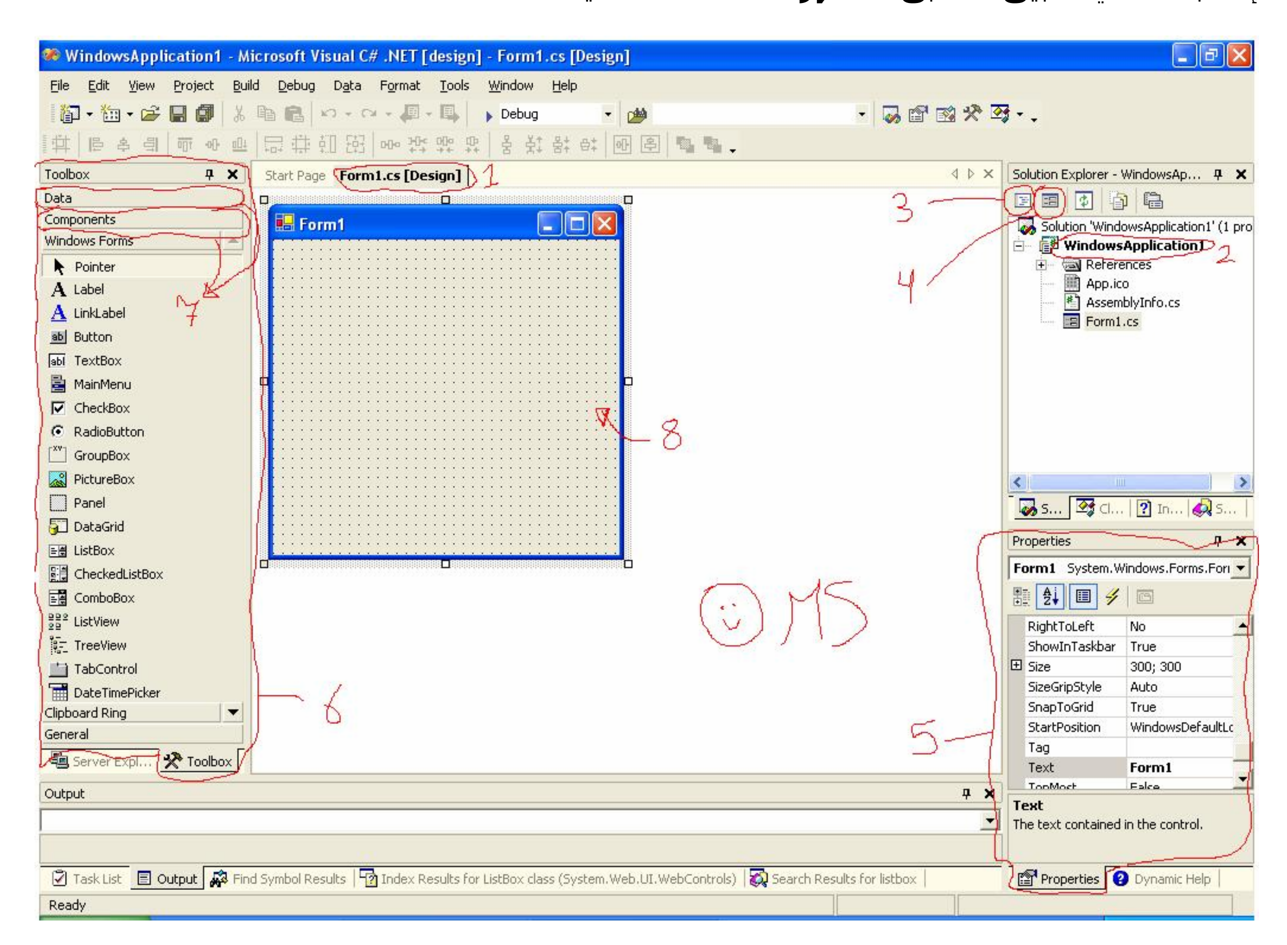
7 – من إسمها ((7)) تعرف أنها مكان منطقة العمل Directory . وبإمكانك تحديد مكان معين بالضغط على الزر الذي يحمل الرقم ((8)) .

8 – إذا ضغط هذا المفتاح ((9))سيفتح لك نافذة مساعدة عن هذه الصفحة فقط .

9 – بإمكانك زيادة الخيارات المستخدمة عندك بإضافة عملك في مجلد جديد على المسار الذي حددته في الخطوة رقم ((7)) وذلك بالضغط على الرقم ((10)) والذي يحمل كلمة More .

10 - وأخيراً قم بعمل OK ((موافق)) لنبدأ مشروعنا اليوم .

إذا نجحت في تطبيق السابق ستظهر معك النافذة التالية :



أنظر إلى النافذة السابقة ولاحظ معي :

1 – قام بفتح نافذة جديدة ((1)) وسماها Form1.cs .

يتكون الإسم من قسمين كالتالي :

القسم الأول يحمل إسم Form1 وهذه تعتبر الكائن الرئيسي في المشروع لأنه سبق لنا أن قلنا أننا نسعى للغة تدعم البرمجة الكائنية OOP ، ولأننا قلنا أنه يجب أن يكون هناك على الأقل كائن واحد يحمل الدالة الرئيسية Main Function ، وبهذه الخطوة إرتقت مايكروسوفت إلى مستوى البرمجة بالكائنات والتي لم تكن موجودة بالإصدارات السابقة .

والقسم الثاني يحمل إسم الإمتداد cs ومعناه C Sharp أي إسم اللغة ، ولاحظ أنه يوجد كلمة [Design] وهي دلالة على أننا في وضع التصميم .
2 –في القسم ((2)) ماذا تلاحظ ؟
المشروع يحتوي على 3 ملفات فقط . ملف للأيقونة الناتجة في المشروع وملف التحويل للغة الأسمبلي والملف الثالث المحتوي على الكود الذي نقوم بكتابته .
ملاحظة سريعة هنا :
حاول فتح أي مشروع غير السي شارب ((ماذا تلاحظ)) ؟
ستقول لي أنه يحتوي عل 3 ملفات أيضاً ، و من خلال هذه الخطوة إستطاعت شركة مايكروسوفت توحيد أنماط جميع لغاتها ضمن باقة الدوت نيت .
3 – هذه الأيقونة لكي تنقلك من نمط التصميم لنمط البرمجة ، أي الكود البحت Code Mode .
4 – لكي تعيدك لنمط التصميم Design Mode .
5 – نافذة جديدة لم تكن في تطبيقات الكونسول Console Application ، وهي نافذة الخصائص Property للكائن المختار في نمط التصميم Design Mode .
6 – الأدوات أو نسميها ال Object أو ال Component ، وهي أي أداة نستعملها للتصميم مثل الأزرار أو صناديق النص أو الإطارات ... إلخ .
7 – تابات للمجموعات Tabs فهي تقسيم من مايكروسوفت وضعت كل الأدوات التي تخص موضوع معين في تاب من هذه التابات ، فمثلاً كل شيء يخص قواعد البيانات وضعته في تاب خاص فيها ، وكل أدوات برمجة الويب (صفحات الإنترنت) وضعته في تاب لوحده ... إلخ .
8 – ساحة العمل والتصميم وتسمى النموذج Form وهي النافذة التي ستظهر عند تنفيذ البرنامج .

- لنأتي الآن إلى منطقة العمل ونشرحها بالتفصيل :
قم بالضغط على الرقم 3 كما في الشكل السابق لتظهر لك النافذة التالية :

Start Page | Form1.cs [Design] | Form1.cs

WindowsApplication1.Form1 ▾　　components ▾

```
1  using System;
2  using System.Drawing;
3  using System.Collections;
4  using System.ComponentModel;
5  using System.Windows.Forms;
6  using System.Data;
7
8  namespace WindowsApplication1
9  {
10     /// <summary>
11     /// Summary description for Form1.
12     /// </summary>
13     public class Form1 : System.Windows.Forms.Form
14     {
15         /// <summary>
16         /// Required designer variable.
17         /// </summary>
18         private System.ComponentModel.Container components = null;
19
20         public Form1()
21         {
22             //
23             // Required for Windows Form Designer support
24             //
25             InitializeComponent();
26
```

((إقرأ الملاحظة بعد الموضوع مباشرة))

لاحظ أنه قام بفتح صفحة خاصة وسماها Form1.cs دون كلمة [Design] ليدلك على أنك في قسم البرمجة Code Mode وليس التصميم .

في السطر رقم (1) جملة `using System;` نستطيع إستنتاج التالي :
* جميع الكلمات الموجودة في منطقة العمل والتي تحمل اللون الأزرق هي كلمات محجوزة Keyword لا نستطيع إستعمالها كمتغيرات .
* تعتبر لغة السي شارب لغة حساسة Case Sensitive يعني أن المتغيرات (Age, AGE, aGE , AgE) كلها متغيرات لا يشبه بعضها بعضاً وتعاملها هذه اللغة كأنها متغيرات مستقلة .

* نهاية كل جملة تحتوي على فاصلة منقوطة ; وهي تعتبر نهاية السطر .

ووظيفة السطر الأول هي إستدعاء مكتبة ((سنتفق على تسميتها namespace)) للتعامل مع المشروع بشكل جيد مثل جمل الإدخال والإخراج وتعادل هذه الجملة بكلمة #include في لغة السي هنا إستدعى مكتبة الـ System ولاحظ أن أول حرف كبير وهذه المكتبة مختصة بالدوال الرئيسية التي تستخدم بكثرة كجمل الإدخال والإخراج وتعادل هذه المكتبة مكتبة الـ iostream.h المستخدمة في لغة السي ++ .
وأيضاً من السطر الأول للسطر السادس ، كلها مكتبات تساعدنا في تصميم تطبيقات الويندوز .

في السطر رقم (8) جملة `namespace WindowsApplication1` نستطيع إستنتاج التالي :
* قام بإنشاء مكتبة تحتوي على المشروع الذي نكتبه الآن .
* تلاحظ أنه يوجد مربع صغير يحتوي على إشارة ناقص (-) ووظيفته إخفاء تفاصيل الكلاس أو الدالة المشار إليها وبعد الضغط عليه يظهر لنا مستطيل يحتوي على ثلاث نقاط (...) إذا حركت الماوس عليه يعطيك محتوى الكلاس أو الدالة المشار إليها كاملاً كشكل ملاحظة Tag بمستطيل أصفر اللون فيعرض لك محتوياتها مهما بلغت من الطول .
والهدف من هذا المربع هو إخفاء دالة أو كلاس سبق لنا أن كتبناها ولا نريد إظهارها أو أننا إنتهينا من كودها بنجاح .

ومعنى هذه الجملة أنه قام بإنشاء مكتبة خاصة والتي تحتوي على الـ Classes الموجودة في مشروعنا الحالي فمثلاً إذا أردنا إستدعاد دالة من الدوال في مشروع آخر ما علينا سوى كتابة إسم المشروع الحالي ثم إتباعه بنقطة ثم إسم الكلاس أو الدالة التي نريد إستعمالها ، طبعاً بعد إضافتها كمرجع Reference إلى مشروعنا.

في السطر رقم (9) القوس المشهور ({) والذي يدل على بداية الدالة أو الكلاس وطبعاً نغلقها بالمثل بإستخدام القوس المثيل (}) كما في السطر رقم (69) .

في السطر رقم (10) `/// <summary>` جملة تعليق ولكنها للغة الـ XML دعها جانباً لن تفيدنا الأن في الوقت الحالي فلها وقتها . لاحظ أنها تحتوي على ثلاث أقواس .

في السطر رقم (12) لاحظ وجود إشارة (-) على العمود وهذا يعني بداية الجملة الأولى في الكلاس أو الدالة Function المعنية .

في السطر رقم (13) تلاحظ وجود إسم الكلاس المستعملة في مشروعنا الحالي وهي `Form1` ، ولكن لاحظ أننا قمنا بالتوارث من المكتبة `System.Windows.Forms.Form` وذلك بإستخدام رمز التوارث (:) أي أن الكلاس الذي نستخدمه هو عبارة عن نموذج Form وهذا يمكننا من إستخدام الكثير من الدوال الموجودة في المكتبة .

في السطر رقم (28) `//` لاحظ وجود قوسين هنا وهما لجمل التعليقات ((لاحظ الفرق في السطر العاشر)) . أي أنها جمل لا معنى لها ، توضّح أو تعلّق على الجمل في تلك السطر .

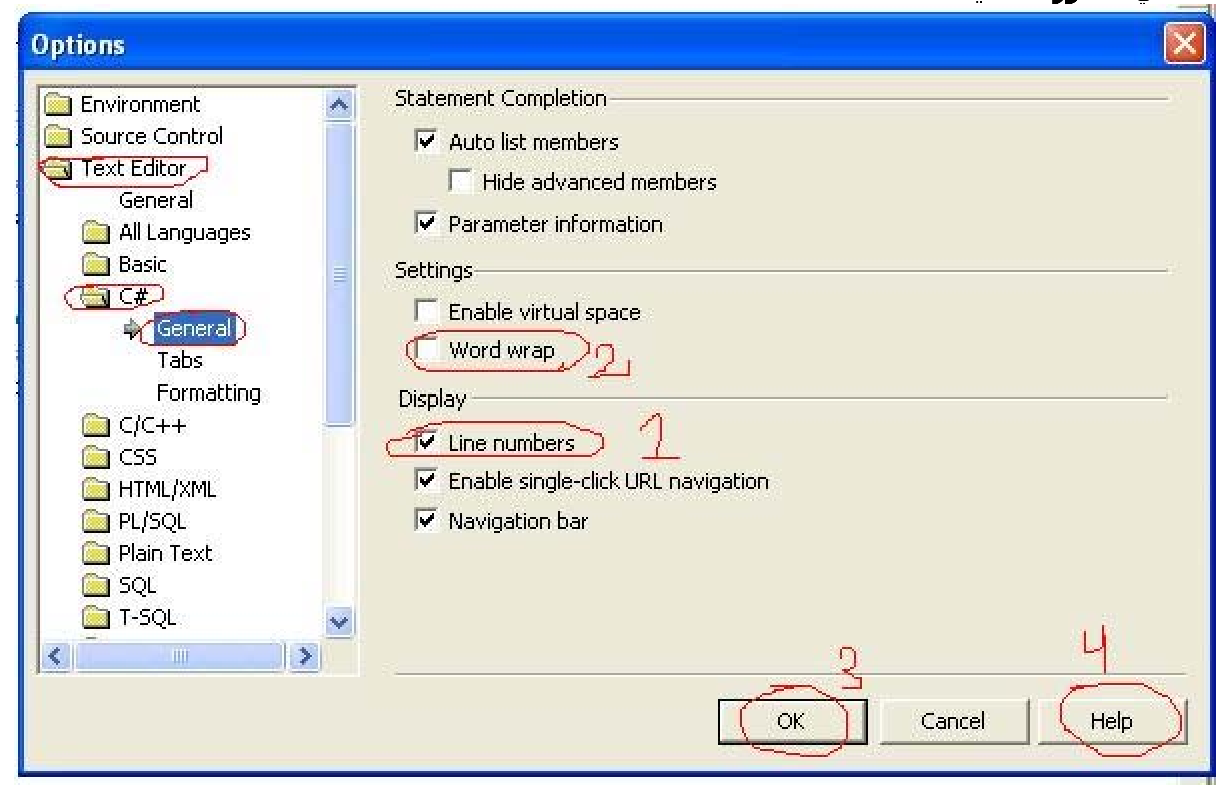
في السطر رقم (63) `[STAThread]` أي كلمة موجودة بين قوسين ((مربعين)) تسمى خاصية Attribute وسنقوم بشرحها لاحقاً .

في السطر رقم (64) `static void Main()` هنا توجد الدالة الرئيسية لمشروعنا .

* static void تحديد نوع الدالة الرئيسية فهي من نوع Void التي تعني أن الدالة لا ترجع أي قيمة وهي من القسم Static من النوع الإستاتيكي .
والنوع الإستاتيكي يمكن شرحه كالتالي : لو أنك عرّفت متغير ما بالنوع الإستاتيكي في دالة معينة في داخل كلاس معين ثم إستدعيت الدالة وأجريت تعديلات على هذا المتغير وخرجت من الدالة فإنه بالوضع الإفتراضي يحذف من الذاكرة ، ولكن هنا ، المتغير الإستاتيكي يقوم بتسجيل نفسه في الذاكرة ما دام البرنامج أو المشروع الذي صممته في وقت التنفيذ . فمثلاً لو عرفت في دالة معينة المتغير X من نوع Int من النوع الإستاتيكي وقمت في سطر تالي بزيادة هذا المتغير بقيمة واحد فإنه كلما قمت بإستدعاء الدالة سينفذ السطر الثاني فقط ويقفز عن السطر الأول (الذي هو جملة التعريف) لأنه موجود في الذاكرة.دعه الآن له وقت سنشرحه بالتفصيل .
* Main لاحظ أن أول حرف كبير .

ملاحظة :
إذا واجهتك أي مشكلة في أي سطر وتريد معرفة المزيد قم بالنقر مرتين مزدوجتين على الجملة ثم قم بالضغط على F1 لظهور نافذة المساعدة بالجملة التي تريد فقط ((يجب أن تمتلك MSDN))

ملاحظة :
إذا لم تحتوي منطقة العمل على أرقام فبإمكانك إضافتها بإتباع المسار التالي :
Tools → Option → Text Editor → C# → General → Line Numbers
كما في الصورة التالية :



قم بوضع ✓ على الرقم ((1))

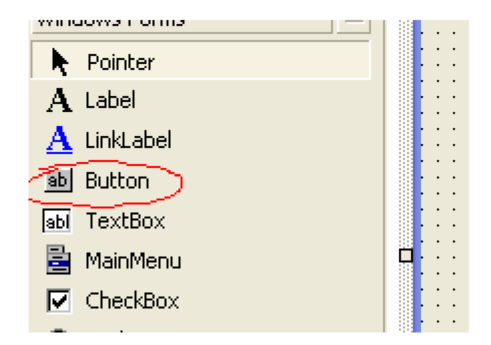
كذلك بإمكانك أن تفعّل خاصية Word Wrap ((2)) والتي تعني أنه بعد حد معين للسطر يقوم بإنزاله إلى سطر جديد ، وبذلك أنت بغنى عن شريط الـ Scroll Bar الأفقي .
أيضاً تستطيع إظهار معلومات أخرى عن هذه النافذة بالضغط على الرقم ((4)) .
الآن إختر كما في النافذة السابقة و إضغط الزر رقم ((3)) OK .

نكون هنا قد شرحنا كود البداية لملف السي شارب بطريقة الـ Windows .

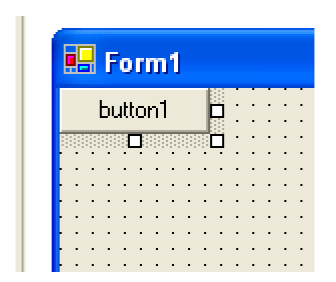
الآن سنقوم بكتابة أول برنامج لنا وهو طباعة جملة معينة ، الآن قم بالإنتقال إلى منطقة التصميم بواسطة الزر :



وقم بإدراج كائن الزر Button .. وذلك بالضغط المزدوج Double Click ، كما في الشكل التالي :



فتقوم اللغة بإدراج الزر Button1 في النموذج Form1 كما في الشكل التالي :



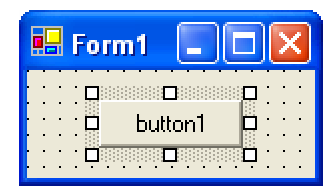
أو يمكنك أنت أن تحدد أبعادة (حجمه وموقعه) وذلك بالضغط مرة واحدة فقط ومن ثم الضغط مع السحب في المنطقة المراد وضع الزر فيها ثم الإفلات .

بعد أن تقوم بإضافة الزر Button1 يمكنك تحريكه بسهولة ووضعه في أي مكان تريده .

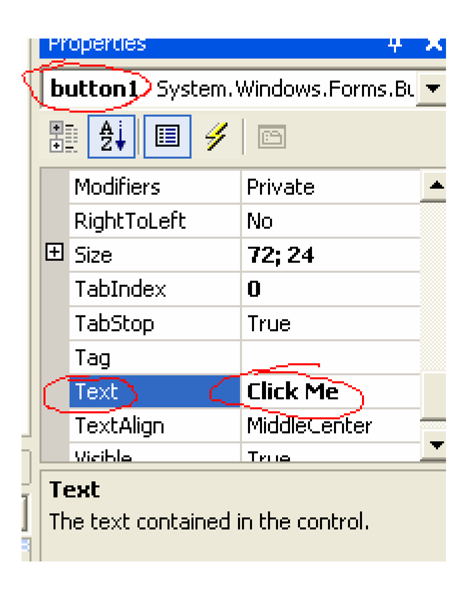
ولكي تضع الزر في وسط الشاشة تماماً قم بإستخدام أدوات التوسيط الطولي والعرضي كما في الشكل التالي :

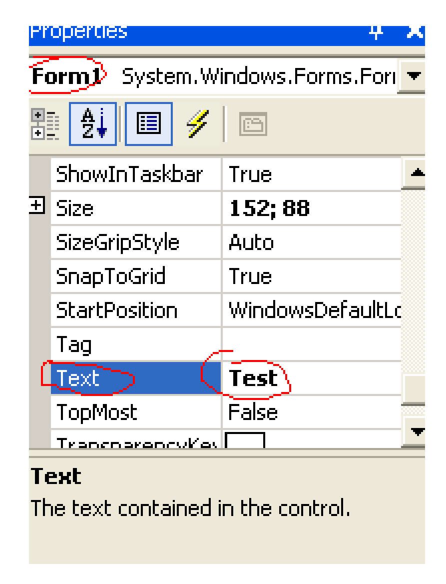


قم بالضغط عليهما بالتوالي .. ولاحظ ما يحدث .
ثم قم بتغيير أبعاد النموذج Form1 والزر كما في الصورة التالية :



وقم الآن بتغيير الأسماء للنموذج والزر وذلك بإختيارهما كل على حدى ، ثم الإنتقال إلى شاشة الخصائص وتغيير الخاصية Text إلى أي الإسم الذي تريد ، وإنتبه هنا الخاصية Text وليس Name . كما في الشاشة التالية :





لتصبح الشاشة كالتالي :



ثم قم بالإنتقال إلى بيئة البرمجة وذلك بالضغط المزدوج على الزد Button1 فيظهر لك الكود التالي :

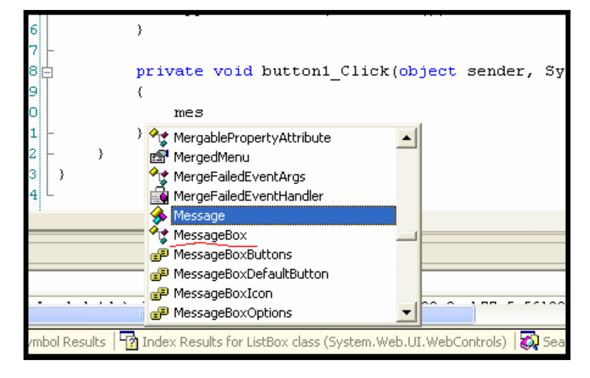
```
        private void button1_Click(object sender, System.EventArgs e)
        {

        }
    }
}
```

- لاحظ أنه قام بإيجاد دالة من النوع الخاص Private وإسمها Button1_Click ولها وسيطين 2 Parameter . ولاحظ أنها تدل على حدث الضغط على زر الماوس .
قم بكتابة السطر التالي :

```
private void button1_Click(object sender, System.EventArgs e)
{
    MessageBox.Show("Welcome To The First Program");
}
```

وهنا تكنيك جميل أنصحك بأن تستخدمه وهو : قم بكتابة الكلمة mes يعني أول ثلاث حروف من الكلمة ثم قم بالضغط على Ctrl + Space فتظهر لك قائمة كما في الشكل التالي :

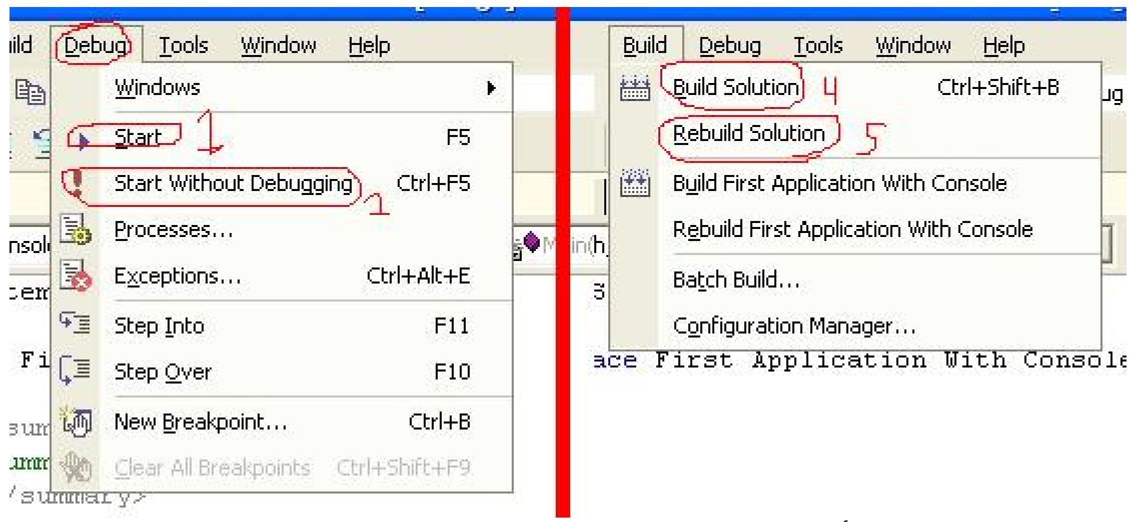


ثم قم بالنزول وإختيار كلمة MessageBox ، ومن أهم خصائص هذه الطريقة أنه لا يشترط أن تكون حافظاً لتركيب الكلمة ، وميزة أخرى فهي تقوم بتعديل الكلمة من حيث الحروف الكبيرة أو الصغيرة إذا أخطأت في كتابتها . فلا تستهين بهذا التكنيك وأنصحك بإستعماله ، فأنا شخصياً لا يخلوا سطر واحد دون أن أستخدم هذا التكنيك و الأمر راجع لك . والآن إلى كيفية تنفيذ البرنامج :

فبعد كتابة الكود قم بالذهاب إلى Debug → Start Without Debugging أو قم بالضغط على الأزرار التالية : F5

لعلك تتساءل : لماذا هذه مع أنه يوجد أوامر كثيرة مثل الـ Debug والـ Start ؟؟

سأقوم بتوضيح الأمر لك . أنظر إلى الصورة التالية :

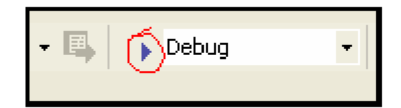


الرقم 1 و 2 : لا فرق بينهما أبداً خلاف تطبيقات الكونسول Console .

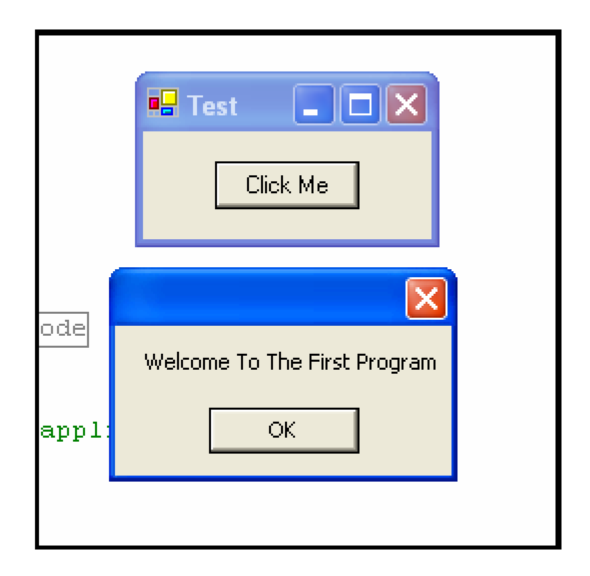
الرقم 4 : يقوم بإنشاء الملفات اللازمة للمشروع بما فيها الملف التنفيذي EXE .

الرقم 5 : يقوم بحذف الملفات السابقة وإنشاء ملفات جديدة حسب التغيرات التي حدثت .

أو يمكنك الضغط الزر التالي :



وعندما تقوم بتنفيذ برنامجك والضغط على الزر Button1 يظهر لك صندوق حوار كالتالي :



مبارك .... لقد أنهيت التطبيق الأول في تطبيقات الويندوز ...